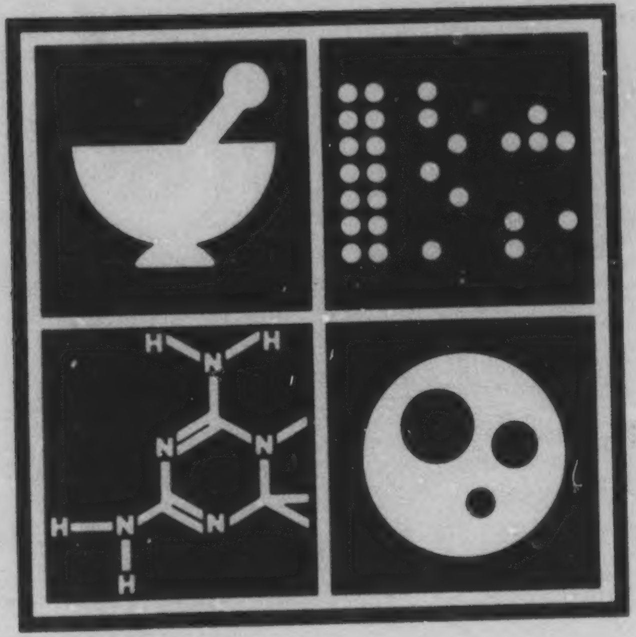


PROPHET

PUBLIC PROCEDURES

U.S. DEPARTMENT OF HEALTH AND HUMAN SERVICES

Public Health Service
National Institutes of Health

P 94/7

PROPHET PUBLIC PROCEDURES

A User's Guide to Public Procedures in
the PROPHET System

DIVISION OF RESEARCH RESOURCES

NATIONAL INSTITUTES OF HEALTH

Prepared by Bolt Beranek and Newman Inc., Cambridge, Mass.
Under Contract # NO1-RR-5-2101
Biomedical Research Technology Program
Division of Research Resources
National Institutes of Health

REVISED SEPTEMBER 1985

NEWARK PUBLIC LIBRARY

MAR 14 1986

U. S. GOVT. PUBS.
DEPOSITORY 376

HE 20.3008:

Introduction 1-1

1. Table Procedures

MAKETABNAMES constructs a list of related table names . . . 1-1
MAKETABS constructs a set of related tables from a template table 1-5
FILLTABS fills data into a set of related tables . . . 1-7
PRINTTABS prints (portions of) related tables 1-9
POOLTABS pools X,Y data from related tables; interpolation and statistics may be performed on pooled results 1-11
DELETETABS deletes a set of related tables 1-21
EDCOL edits data in a column of a table 1-23
TABLE2FILE reads data from a table into a formatted ASCII file 1-27

2. Graph Procedures

DTOGPOINT finds graph coordinates of tablet-indicated points 2-1
EDITPOINTS edits the points of a graph by using the graphics tablet. 2-3
DIGITIZE creates a graph from input traced on the graphics tablet. 2-11
GRAPH3D specifies data to be displayed in three dimensions by DRAW3D or MULT3D. 2-21
DRAW3D displays and manipulates data in three dimensions 2-29
MULT3D displays and manipulates several sets of data in three dimensions 2-43
BARGRAPH constructs a bar graph from percentile data . 2-47

READEX FICHE best copy available

XFROMY computes X values for indicated Y values of a graph (optionally accepts Y values of the form ED50) 2-53

3. Numerical Procedures

ROOT finds real and complex roots of a polynomial . 3-1

EIGEN computes eigenvalues and eigenvectors of a real symmetric matrix 3-5

INTERPOLATE performs cubic spline interpolation 3-9

SMOOTH smooths experimental data. 3-17

PLANIM calculates the area within a closed curve . . 3-21

INTEGRAL finds the area under or bounded by a curve . . 3-23

TRAP finds the area under a curve using the trapezoidal rule 3-27

STEPAREA finds the incremental area under an exponential decay curve using the cubic spline method . . 3-31

4. Modelling Procedures

PROBIT performs probit analysis of dose-response data for estimating ED50 4-1

AGONIST analyzes dose-response curves to estimate full or partial agonist dissociation constant . 4-7

ANTAGONIST analyzes dose-response curves to estimate competitive antagonist dissociation constant . 4-17

LOGISTIC performs iterative least-squares fit of a logistic function 4-27

ABSORB computes and plots absorption profiles . . . 4-35

DIFFEQ fits and integrates systems of differential equations. 4-43

DRUGFUN fits parameters of standard pharmacokinetic models. 4-63

DRUGAUC uses DRUGFUN results to estimate the area under a concentration-time curve plus related measures. 4-81

MKMODEL performs simulation or curve fitting of one or more functions 4-89

GOFIT displays goodness-of-fit measures for models fitted with DRUGFUN or MKMODEL. 4-127

GRAFIT graphically displays the results of DRUGFUN or MKMODEL 4-131

MODELTEST compares models used to fit curves to a common set of observations 4-139

CELLSIM simulates patterns of growth in cell populations 4-147

5. Utility Procedures

COMPSAVED recompiles saved procedures 5-1

FINDARCHIVED performs census of archived tables on an archive pack. 5-3

DELETEARCHIVED sets up a batch job that deletes tables on an archive pack using results of FINDARCHIVED. 5-9

MAKEFORMATTAB creates a format table to be used by READFILE 5-13

READFILE transfers data from local computers to PROPHET tables 5-19

READTEXT transfers data from local computers to PROPHET TEXT variables 5-29

P2MLAB transfers data from PROPHET to MLAB 5-31

MLAB2P transfers data from MLAB to PROPHET 5-35

P2BMDP transfers PROPHET data to BMDP, creates a control file, and runs BMDP programs 5-39

6. Fun Procedures

HAPPY cheers you up 6-1

POLYGON a many-sided procedure 6-3

SPIRO makes pretty pictures 6-5

LETTERS your own personal calligrapher. 6-9

7. Appendices

A. Suggested Guidelines for Writing a Public Procedure . . . A-1

B. Documenting a Public Procedure B-1

C. Using the Public Table "PUBSAVED" to Develop a Public Procedure C-1

D. Incorporating Public Procedures into your PL/PROPHET Programs D-1

8. Index

Introduction

The PROPHET Public Procedures Notebook is designed to provide a comprehensive and integrated description of the scope and use of helpful, general-purpose procedures that are available to all PROPHET users.

Since the last edition of PROPHET Public Procedures, the PROPHET user community has expanded both in numbers and in breadth of interests. This is reflected in the increased number of public procedures described here, as well as in the diversity of their applications. The entire user community is indebted to the users who have so generously shared their programming skills by making their procedures public.

Please note that PROPHET public procedures relevant to statistical applications are described in the PROPHET Statistics manual. Similarly, public procedures designed to be used in conjunction with the PROPHET molecule facilities are described in PROPHET Molecules.

Organization of this Notebook

This notebook consists of six major sections:

Section 1 consists of procedures useful for managing and processing PROPHET tables.

Section 2 describes procedures that construct and modify PROPHET graphs.

Section 3 includes procedures that perform numerical computations.

Section 4 describes modelling procedures that perform curve fitting and simulation for various applications.

Section 5 consists of "utility" procedures. It contains tools for data management, procedures for transferring files, and interfaces between PROPHET and other programs.

Section 6 consists of procedures that are fun to use.

In addition, appendices to the notebook provide handy supplemental information concerning public procedures:

Appendix A presents helpful tools and suggestions for writing public procedures.

Appendix B lists guidelines for documenting public procedures.

Appendix C introduces the public table PUBSAVED and offers suggestions for using it.

Appendix D describes how to use public procedures in PL/PROPHET programs.

Format of Procedure Descriptions

Each procedure is described in a consistent and concise manner. The description begins with the principal name of the procedure, the author(s) of the procedure, and a "summary box".

Each summary box tells you briefly what the procedure does and how to use it by providing the following information:

Summary: a brief explanation of what the procedure does

Calling Sequence: the PROPHET commands required to use the principal procedure. (NOTE THAT YOU MUST END EVERY COMMAND BY PRESSING THE "GO" KEY. This will be indicated by "<GO>".)

Entries: alternative entry points associated with the procedure (see below). Be sure to first RAC (Restore and Compile) the principal procedure before using an entry

Citation: a suggested format for citing the public procedure

Questions or Comments: the person to whom you should direct any questions about the procedure, along with information on how to contact that person

Things to Remember: important requirements and caveats associated with using the procedure

References are also included where appropriate.

Entries to Procedures

You can use almost all procedures in either of two different modes: an interactive mode and a "quiet" mode. Typically, when you call a procedure using its principal name, e.g., CALL INTERPOLATE, you enter a dialogue in which you are prompted for any required input and are asked which results you want to see.

When you call a procedure using its "quiet" entry, e.g., CALL INTERPOLATEQT(<list-of-arguments>), you do not enter a dialogue. No questions are asked, and no results are presented. Instead, quiet entries require arguments that direct the behavior of the procedure and save the results without your intervention.

Quiet entries are particularly useful within PL/PROPHET procedures you write and for batch jobs (see Appendix C), since your procedure "knows" the values of the necessary arguments and therefore frees you from the need to supply them interactively.

By convention, a quiet entry has the suffix QT. Thus, INTERPOLATE and INTERPOLATEQT are functionally equivalent, but behave very differently when you use them.

Empty Arguments

Occasionally, you will encounter a description that refers to an EMPTY argument, that is, an argument with no value. If you ever use an EMPTY argument, be sure not to omit it. For example, the quiet entry P2MLABQT of the public procedure P2MLAB requires two arguments:

- the text name of a piece of data you wish to save in a file that MLAB can read
- the text name of a file in which to save the data (if EMPTY, the data is automatically saved in a file named MLASAV)

Thus, to save a table named DAT in the file MLASAV, you would type the command:

```
CALL P2MLABQT ('DAT', ) <GO>
```

You must include the comma in the call to P2MLABQT because it indicates the presence of a second argument. If you specify no value for this argument, it is treated as an EMPTY argument. If you omitted the comma, the command would abort and result in an error message:

```
WRONG NUMBER OF ARGUMENTS.
```

Procedures That Return Results

Many procedures and entries <u>return</u> values. For instance, the public procedure TRAP requires a data table as an argument, and it uses the trapezoidal rule to compute and return the area under a curve. The procedure uses columns of the data table for X and Y values. This is described in the summary box of TRAP as:

```
float-var = TRAP(data-tab)
                 <table>
```

Notice that the procedure appears on the right-hand side of the assignment statement because it is returning its result. The T on the left hand side serves as a placeholder for the returned result.

If your data table were named XYDAT, then to use TRAP to obtain the trapezoidal area and save it in T, you would type:

```
T = TRAP(XYDAT) <GO>
```

If you subsequently gave the command

```
TYPE T <GO>
```

PROPHET would type the result on the terminal.

If instead, you had typed the command

```
CALL TRAP(XYDAT) <GO>
```

then TRAP would be called with XYDAT as an argument, and would consequently compute and return the area, but nothing would be done with the returned value. It is almost the same as simply typing

3.14 <GO>

since the call to TRAP evaluates to a float number.

Before you use a public procedure or entry, be sure you understand whether it returns a value. If it does, be sure to save or type the returned result.

Using This Notebook

If you are about to use a particular public procedure for the first time, you should carefully read the detailed documentation and consult the examples that follow the summary box. You should use the summary box only as a quick reference to the procedure.

Numerous examples are included throughout this notebook. These examples include samples of required input data, interactions with the procedure, and many examples of the results produced.

Usually, sample public tables are available for procedures requiring input tables. The names of the public tables are shown in the example so you can run the example yourself. These public tables are usually archived, so you may have to dearchive them before running the procedure.

It is important that the operation of a public procedure always be reflected in its documentation. Consequently, you should never change a public procedure without consulting the BBN staff. Of course, at any time you can make a private copy of any public procedure and change it to meet your own specific needs.

Making Your Procedure Public

If you have written a procedure that you think other PROPHET users might find useful, please notify us. Consult Appendices A and B for some guidelines for writing and documenting public procedures. Appendix C describes the public table PUBSAVED, which you can use to develop a procedure by letting other PROPHET users run it and provide feedback.

1. Table
Procedures

1. Table
Procedures

1. Table Procedures

MAKETABNAMES	constructs a list of related table names . . .	1-1
MAKETABS	constructs a set of related tables from a template table	1-5
FILLTABS	fills data into a set of related tables . . .	1-7
PRINTTABS	prints (portions of) related tables	1-9
POOLTABS	pools X,Y data from related tables; interpolation and statistics may be performed on pooled results	1-11
DELETETABS	deletes a set of related tables	1-21
EDCOL	edits data in a column of a table	1-23
TABLE2FILE	reads data from a table into a formatted ASCII file	1-27

MAKETABNAMES

by
Nick Holford
Division of Clinical Pharmacology
University of California, San Francisco
and
Harold Perry
Bolt Beranek and Newman Inc.

Summary

MAKETABNAMES enables you to construct and save a TEXT ARRAY name list of table names composed of as many as four different codes.

Calling Sequence

```
RAC PUBLIC MAKETABNAMES <GO>
CALL MAKETABNAMES <GO>
```

Entries

```
CALL MAKETABNAMESQT(code1-list, code2-list, code3-list, code4-list,
                      <text>       <text>       <text>      <text>
                                                  or
                                                <fixed>

                    array-name,  save-it)
                      <text>      <bool>
```

Citation

No citation necessary.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things to Remember

For MAKETABNAMES to construct an array of legal PROPHET table names, the first non-empty code-list must consist entirely of codes of type TEXT.

To facilitate management of the tables, you can use the constructed TEXT ARRAY of table names in related public procedures, including MAKETABS and PRINTTABS. You are encouraged to use this name list in ARCHIVE and DEARCHIVE commands.

The public procedure MAKETABNAMES constructs a TEXT ARRAY of table names. It creates each name by joining together as many as four codes, which can be any length. A table name composed in such a manner clearly indicates the type of data contained in that table.

For example, suppose you want to record data for five subjects who receive quinidine by intravenous and oral routes. If the data are collected from plasma and urine, you can use MAKETABNAMES to construct a list of tables from the following lists of codes:

code1-list:	P,U	(denotes whether data is derived from plasma or urine)
code2-list:	IV,PO	(denotes whether quinidine is administered intravenously or orally)
code3-list:	1,2,...,5	(denotes subject)

By combining these codes, MAKETABNAMES would produce twenty table names:

PIV1, PIV2, PIV3, PIV4, PIV5, PPO1, PPO2, PPO3, PPO4, PPO5
UIV1, UIV2, UIV3, UIV4, UIV5, UPO1, UPO2, UPO3, UPO4, UPO5

Using MAKETABNAMES

When you call MAKETABNAMES, the procedure prompts you to enter four lists of codes to be combined into table names. If you want to specify fewer than four lists of codes, simply press <GO> in response to any of the procedure's code-list prompts. The codes within each list may be separated by commas, spaces, or carriage returns.

The first non-empty code-list must begin with an alphabetic character; otherwise, the procedure will create illegal table names. MAKETABNAMES asks whether you want to use a list of increasing consecutive integers (starting with 1) for the third code-list. If you do, you need only specify the last number in the list.

After you specify the code-lists, MAKETABNAMES displays all of the table names it has created. It then asks you to supply a name for the TEXT ARRAY in which these table names are stored. It also gives you the option of saving this array in your PRIVATE SAVED table.

Consult Figure 1 for illustration of your interaction with MAKETABNAMES.

```
>RAC PUBLIC MAKETABNAMES<
>CALL MAKETABNAMES<
ENTER CODE LISTS SEPARATED BY COMMAS, SPACES OR CARRIAGE RETURNS
(SINGLE CODE LISTS OR EMPTY CODE LISTS ARE OK)

CODE1-LIST?P,U<
CODE2-LIST?IV,PO<
DO YOU WANT CONSECUTIVE NUMBERS FROM 1 ONWARDS
TO BE USED AS THE CODE3-LIST?Y<
LAST NUMBER?5<
CODE4-LIST?<
NAMES ...

 PIV1 PIV2 PIV3 PIV4 PIV5 PPO1 PPO2 PPO3 PPO4 PPO5 UIV1 UIV2 UIV3 UIV-
4 UIV5 UPO1 UPO2 UPO3 UPO4 UPO5


WHAT NAME DO YOU WANT FOR THE LIST OF TABLE NAMES?STUDY1<
DO YOU WANT TO SAVE THE LIST OF TABLE NAMES?Y<
STUDY1 SAVED IN PRIVATE SAVED
>
```

Figure 1. Using MAKETABNAMES.

Quiet Entry MAKETABNAMESQT

MAKETABNAMES also has a quiet entry, MAKETABNAMESQT, which requires the arguments shown in Table 1.

Table 1. MAKETABNAMESQT Arguments.

Argument	Type	Value
code1-list	<text>	List of choices for code 1, separated by commas. (If none, it should be empty.)
code2-list	<text>	List of choices for code 2, separated by commas. (If none, it should be empty.)
code3-list	<text> or <fixed>	If you want code3-list to take on the values 1, ..., n for some n, then this argument may have some fixed value. Otherwise, it can be general choices separated by commas. (If none, it should be empty.)
code4-list	<text>	List of choices for code 4, separated by commas. (If none, it should be empty.)
array-name	<text>	Desired name of constructed TEXT ARRAY of related table names.
save-it	<bool>	If TRUE, the list of tables is automatically saved under array-name.

Thus, to use MAKETABNAMESQT to construct the array in the previous example, you would give the command:

```
CALL MAKETABNAMESQT('P,U', 'IV,PO', 5,
                    'STUDY1', TRUE) <GO>
```

Note that the resulting TEXT ARRAY name list would be saved in the PRIVATE SAVED table as STUDY1.

MAKETABS

by
Nick Holford
Division of Clinical Pharmacology
University of California, San Francisco
and
Harold Perry
Bolt Beranek and Newman Inc.

Summary

MAKETABS uses a template table to construct a set of tables whose names are contained in a TEXT ARRAY.

Calling Sequence

```
RAC PUBLIC MAKETABS <GO>
CALL MAKETABS <GO>
```

Entry

```
CALL MAKETABSQT(tab-names,  template-table)
               <text-array>     <table>
```

Citation

No citation necessary.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things to Remember

MAKETABS ignores entries in the TEXT ARRAY of table names that already exist as tables.

Be sure you have a template table before calling MAKETABS.

The public procedure MAKETABS constructs a set of tables whose names are contained in a TEXT ARRAY.

Before calling MAKETABS, be sure you have constructed a table that can be used as a template; the set of tables that MAKETABS constructs will be identical to this template except for the name. For example, a template table for the example from the public procedure MAKETABNAMES might include time values entered in column 1 and the heading of column 2 set to QUINIDINE CONCENTRATION. Note that the time values would be included in column 1 of every table you construct, whereas column 2 would be empty because each table would have different quinidine data.

When you call MAKETABS, the procedure asks you for the name of the TEXT ARRAY containing the list of tables you would like to construct. It then prompts you for the name of your template table. Note that the name of this template table may already be contained in the list of names. MAKETABS automatically ignores any entry that is already a table.

```
>RAC PUBLIC MAKETABS<
>CALL MAKETABS<
NAME OF TEXT ARRAY THAT CONTAINS TABLES TO BE MADE?STUDY1<
NAME OF TABLE TO BE USED AS A TEMPLATE?PIV1<
PIV1 ALREADY EXISTS
[CONSTRUCTING PPO1......DONE]
[CONSTRUCTING UIV1......DONE]
[CONSTRUCTING UPO1......DONE]
>
```

Figure 1. Using MAKETABS.

Quiet Entry
MAKETABSQT

MAKETABS also has a quiet entry, MAKETABSQT. It requires the following arguments:

tab-names <text array>	list of names of tables to be made
template-table <table>	template from which the tables specified in tab-names are to be made

FILLTABS

by
Nick Holford
Division of Clinical Pharmacology
University of California, San Francisco
and
Harold Perry
Bolt Beranek and Newman Inc.

Summary

FILLTABS enables you to fill a set of tables with data.

Calling Sequence

```
RAC PUBLIC FILLTABS <GO>
CALL FILLTABS <GO>
```

Citation

No citation necessary.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things to Remember

Use the FILL ALL command to add new data or modify old data. Use FILL to enter only new data.

FILLTABS enables you to enter and change data in a set of related tables whose names are specified in a TEXT ARRAY name list.

When you call FILLTABS, the procedure asks for the name of the TEXT ARRAY containing the names of the tables you wish to fill. It next asks whether you would like to fill every table within the array. If not, answer NO; FILLTABS then lets you indicate the tables you want to fill.

Finally, FILLTABS asks whether you would like to FILL or FILL ALL the specified tables. FILL enables you to enter only new data into each table, whereas FILL ALL lets you both enter new data and modify old data. Consult the *PROPHET User's Manual* for complete descriptions of these commands.

```
>RAC PUBLIC FILLTABS<
>CALL FILLTABS<
NAME OF TEXT ARRAY THAT CONTAINS TABLES TO BE FILLED?TS<
DO YOU WANT TO FILL EVERY TABLE SPECIFIED IN TS?NO<

                   4R X 1C

                 | 1
              ---+-----
              1  | PIV1
              2  | PPO1
              3  | UIV1
              4  | UPO1

TYPE THE INDICES OF THOSE TABLES YOU WANT TO FILL
(SEPARATED BY COMMAS OR SPACES)?1,3,4<

FOR EACH SPECIFIED TABLE, DO YOU WANT TO:
1       FILL (ENTER NEW DATA) OR
2       FILL ALL (ENTER NEW DATA AND UPDATE OLD DATA)?2<

REMEMBER TO MOVE (:M ROW,COL) TO THE COLUMN YOU WISH TO FILL


FILLING TABLE PIV1

        <FILL TABLE PIV1>

FILLING TABLE UIV1

        <FILL TABLE UIV1>

FILLING TABLE UPO1

        <FILL TABLE UPO1>
```

Figure 1. Using FILLTABS.

PRINTTABS

by
Nick Holford
Division of Clinical Pharmacology
University of California, San Francisco
and
Harold Perry
Bolt Beranek and Newman Inc.

Summary

PRINTTABS prints a set of tables. It can print the tables in their entirety, or it can print only those portions of the tables that you specify with a single FOR-clause.

Calling Sequence

```
RAC PUBLIC PRINTTABS <GO>
CALL PRINTTABS <GO>
```

Citation

No citation necessary.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things To Remember

The public procedure PRINTTABS prints tables contained in a name list of tables. The name list must be a TEXT ARRAY.

When you call PRINTTABS, the procedure asks for the name of the TEXT ARRAY name list containing the names of the tables you wish to print. It next asks whether you would like to print every table within the array. If not, type NO; PRINTTABS then lets you indicate the tables you want to print.

If you would like to print only certain portions of the tables, you can now enter a FOR-clause. PRINTTABS applies this FOR-clause to every table it prints.

PRINTTAB's final question is:

NUMBER OF TABLES TO BE PRINTED PER PAGE?

If you know how many tables (or FOR-clause-specified portions of tables) will fit on the terminal screen at one time, you can enter that number to display and print that number of tables simultaneously. Otherwise, simply press the <GO> key; the default is one table or table portion per page (screen).

```
<The following example illustrates how PRINTTABS may be
used to print the first 3 columns from a subset of tables.
Each printed hard copy will contain 2 table portions.>


>RAC PUBLIC PRINTTABS<
>CALL PRINTTABS<
NAME OF TEXT ARRAY THAT CONTAINS TABLES TO BE PRINTED?TABS<
DO YOU WANT TO PRINT EVERY TABLE SPECIFIED IN TABS?NO<

TYPE THE INDICES OF THOSE TABLES YOU WANT TO PRINT
(SEPARATED BY COMMAS OR SPACES)?1,2,5,6<

ENTER "FOR CLAUSE" TO BE USED WITH EACH PRINT COMMAND
(IF NONE HIT <GO>)?FOR COLUMNS 1 TO 3<

NUMBER OF TABLES TO BE PRINTED PER PAGE?2<

<The indicated tables will then be printed automatically.>
```

Figure 1. Using PRINTTABS.

POOLTABS

by Nick Holford
Division of Clinical Pharmacology
University of California, San Francisco

Summary

POOLTABS enables you to pool X and Y data from related tables into a single table. POOLTABS can use MLAB to interpolate the Y data using a new set of X values. It also enables you to obtain statistical measures of the pooled results.

Calling Sequence

```
RAC PUBLIC POOLTABS <GO>
CALL POOLTABS <GO>
```

Entry

```
CALL POOLTABSQT(list-of-tables,  X-col,  Y-col, source-id,
                <text array>   <fixed> <fixed>   <text>

                what-to-do,  results-table, extrap)
                  <text>       <table>      <bool>
```

Citation

No citation necessary.

Questions or Comments

Send a message to 'HOLFORD' using PROPHET.

Things to Remember

Before you call POOLTABS, you must store the names of the tables you want to pool in a TEXT ARRAY name list.

The column structures of the tables you want to pool must be identical; that is, the X data must be in the same column in each table, as must the Y data.

POOLTABS ignores empty values in the tables of data it pools.

Introduction

POOLTABS pools sets of X,Y data from related tables into a single results-table. The names of these related tables must be contained in a TEXT ARRAY name list. (You can construct a name list with the public procedure MAKETABNAMES.)

If the sets of X,Y data do not all involve a common set of X values, you can use POOLTABS to perform cubic spline interpolation for each set of X,Y data using a fixed set of X values. POOLTABS also gives you the option of computing statistical measures of your pooled data.

Interacting with POOLTABS

To use the procedure, type:

```
CALL POOLTABS <GO>
```

Once called, POOLTABS asks for the name of a TEXT ARRAY containing a list of table names.

The procedure then asks you to specify the column of each table that contains X data. Specify just one column number; remember, the X data must be in the same column in every table. POOLTABS will later ask you to supply a new set of X values for which you want interpolated Y values.

If your tables contain a common set of X values (or, equivalently, if they contain _no_ X values, as do the tables in Figure 1 below), you do not need to perform an interpolation. Simply type CTRL-E in response to POOLTABS' prompt.

You must next specify the number of the column in which Y data are stored in each of the tables you wish to pool. Like the X data, the Y values must be in the same column in every table.

POOLTABS then asks you to supply a "source identification code". This code is optional and can be any combination of characters. If you specify a code, POOLTABS includes it in the heading of every Y data column in the results-table. If you do not want to use a source code, simply press <GO> in response to the procedure's prompt.

You now have the option of computing a set of statistical measures for your npooled data. POOLTABS computes each measure across each row of the pooled results-table, so the statistics correspond to relationships between all the Y data for a single X value. Depending on the answer you give to POOLTAB's prompt, your results-table will include both the pooled data and the statistical measures, or only the statistics, or only the pooled data. Type:

BOTH <GO>	for the pooled data along with the row-wise statistics
STATS <GO>	for just the row-wise statistics
DATA <GO>	for just the pooled data

If you request statistics of the pooled results, POOLTABS applies the PROPHET command STATS to the rows of the pooled data. Thus, by using the SELECT STATS command before calling POOLTABS, you can tailor the set of statistics that will be computed. (Consult the PROPHET Statistics Manual for descriptions of the STATS and SELECT STATS commands.)

POOLTABS next asks you for the name of your results-table. This table must be available in your workarea before you call the procedure. If you want POOLTABS to perform interpolation, column 1 must contain the new X values for interpolation.

The pooled data and/or the statistical measures are now added to the results-table. POOLTABS ignores empty values in the pooled tables.

If you are performing interpolation, the range of X values in the destination table may be outside the range of X values in the source data. The procedure gives you the option of extrapolating the new values for these X values depending upon your needs.

Quiet Entry
POOLTABSQT

A quiet entry, POOLTABSQT, is also available. Once called, POOLTABSQT asks no questions. It requires the arguments shown in Table 1.

Table 1. POOLTABSQT Arguments.

Argument	Type	Value
list-of-tabs	<text array>	list of names of related tables
X-col	<fixed>	column of tables with X data; should be empty only if you are not performing interpolation
Y-col	<fixed>	column of tables with Y data
source-id	<text>	any identification code that you want included in the Y column headings of the results-table
what-to-do	<text>	'BOTH', 'STATS', or 'DATA' (see above for a description of each)
results-table	<table>	table to receive pooled data; if the X-col argument is not empty, it should have X-values in column 1
extrap	<bool>	extrapolation of outer range X-values if TRUE, no extrapolation of outer X-values if FALSE

Examples

In the first example below, the four related tables contain only Y data; thus, they all implicitly involve a common set of X values (Figure 1). The common X values are explicitly specified in the results-table, RESTAB (Figure 2).

PONDPH1: PH LEVEL 8R X 1C

	1 POND 1
1	7.68
2	7.7
3	7.72
4	7.73
5	7.73
6	7.76
7	7.78
8	7.8

PONDPH3: PH LEVEL 8R X 1C

	1 POND 3
1	7.74
2	7.75
3	7.77
4	7.78
5	7.8
6	7.81
7	7.81
8	

PONDPH2: PH LEVEL 8R X 1C

	1 POND 2
1	7.69
2	7.7
3	7.71
4	7.75
5	7.74
6	7.74
7	7.78
8	7.81

PONDPH4: PH LEVEL 8R X 1C

	1 POND 4
1	7.71
2	7.71
3	7.74
4	7.78
5	7.84
6	7.85
7	7.87
8	7.91

Figure 1. Related tables containing only Y data.

RESTAB 8R X 0C

0 TIME
WEEK 1
WEEK 2
WEEK 3
WEEK 4
WEEK 5
WEEK 6
WEEK 7
WEEK 8

Figure 2. X values in the results-table before calling POOLTABS.

Figure 3 illustrates your interaction with POOLTABS. Note that, in this example, you neither perform interpolation nor specify a source identification code. TS is the name of the TEXT ARRAY containing the names of the four related tables.

```
>RAC PUBLIC POOLTABS<
>CALL POOLTABS<
NAME OF TEXT ARRAY WITH TABLES TO BE POOLED?TS<
COLUMN OF TABLES NAMED IN TS WITH X DATA
(IF NOT PERFORMING INTERPOLATION, TYPE 'CTRL-E')?E^<
COLUMN OF TABLES IN TS WITH Y DATA?1<
SOURCE IDENTIFICATION CODE
(IF NONE, HIT <GO>)?<
WHAT DO YOU WANT TO DO. TYPE

        'BOTH'          FOR BOTH POOLED DATA AND STATISTICS
        'STATS'         FOR STATISTICS ON POOLED DATA ONLY
        'DATA'          FOR POOLED DATA ONLY

?BOTH<
 NAME OF RESULTS TABLE ?RESTAB< 10 COLUMNS ADDED
```

Figure 3. Using POOLTABS to pool data with common X values.

Figure 4 shows the results-table after POOLTABS has pooled your data and computed the statistics for them.

RESTAB 8R X 10C

0 TIME	1 PONDPH1	2 PONDPH2	3 PONDPH3	4 PONDPH4	5 -N-	6 -SUM-	7 -MEAN-	8 -VARIANCE-	9 -STDEV-	10 -CI95-
1. WEEK 1	7.68	7.69	7.74	7.71	4.	30.82	7.705	.0006987254	.02643341	.04202913
2. WEEK 2	7.7	7.7	7.75	7.71	4.	30.86	7.715	.0005658468	.02378753	.03782218
3. WEEK 3	7.72	7.71	7.77	7.74	4.	30.94	7.735	.0006999969	.02645746	.04206735
4. WEEK 4	7.73	7.75	7.78	7.79	4.	31.05	7.7625	.0007591248	.02755222	.04380803
5. WEEK 5	7.73	7.74	7.8	7.84	4.	31.11	7.7775	.00269254	.0518897	.08250462
6. WEEK 6	7.76	7.74	7.81	7.85	4.	31.16	7.79	.002466838	.04966727	.07897096
7. WEEK 7	7.78	7.78	7.81	7.87	4.	31.24	7.81	.001799901	.04242524	.06745614
8. WEEK 8	7.8	7.81		7.91	3.	23.52	7.84	.00370121	.06083757	.1510357

Figure 4. Results-table after using POOLTABS.

The next example illustrates how you can use POOLTABS to interpolate your Y data using a new set of X values. The three related tables are shown in Figure 5; X values are in column 2, and Y values are in column 3. Figure 6 shows the results-table containing the new X values.

PIV1A 8R X 3C

	1 SCHEDULE TIME	2 TIME ACTUAL	3 PLASMA CONC
1	0.	0.	10.
2	5.	4.	9.54851
3	10.	9.	9.012704
4	15.	14.	8.506964
5	30.	34.	6.752313
6	60.	55.	5.29803
7	120.	121.	2.472019
8	240.	237.	.6474169

PIV2A 8R X 3C

	1 SCHEDULE TIME	2 TIME ACTUAL	3 PLASMA CONC
1	0.	1.	9.885164
2	5.		
3	10.	13.	8.605789
4	15.	19.	8.029603
5	30.	36.	6.598122
6	60.	75.	4.205256
7	120.	100.	3.150575
8	240.	200.	.9926125

PIV3A 8R X 3C

	1 SCHEDULE TIME	2 TIME ACTUAL	3 PLASMA CONC
1	0.	0.	10.
2	5.	6.	9.330467
3	10.	8.	9.117404
4	15.	18.	8.122883
5	30.	27.	7.320913
6	60.	61.	4.94331
7	120.		
8	240.	270.	.4422345

Figure 5. Related tables containing Y values and differing X values.

RESTAB 8R X 1C

	1 SCHEDULE TIME
1	0.
2	5.
3	10.
4	15.
5	30.
6	60.
7	120.
8	240.

Figure 6. Results-table containing new X values for interpolation.

Figure 7 illustrates your interaction with the procedure. This time, you specify a source identification code, STUDY1. As in the previous example, the related tables' names are stored in a TEXT ARRAY name list called TS. The results are shown in Figure 8.

```
>RAC PUBLIC POOLTABS<
>CALL POOLTABS<
NAME OF TEXT ARRAY WITH TABLES TO BE POOLED?TS<
COLUMN OF TABLES NAMED IN TS WITH X DATA
(IF NOT PERFORMING INTERPOLATION, TYPE 'CTRL-E')?2<
COLUMN OF TABLES NAMED IN TS WITH Y DATA?3<
SOURCE IDENTIFICATION CODE
(IF NONE, HIT <GO>)?STUDY1<
SHOULD EXTRAPOLATION BE ALLOWED?YES<

WHAT DO YOU WANT TO DO. TYPE

         'BOTH'          FOR BOTH POOLED DATA AND STATISTICS
         'STATS'         FOR STATISTICS ON POOLED DATA ONLY
         'DATA'          FOR POOLED DATA ONLY

?BOTH<
NAME OF RESULTS TABLE?RESTAB<  PIV1A PIV2A PIV3A  9 COLUMNS ADDED
>
```

Figure 7. Using POOLTABS to interpolate a new set of X data.

RESTAB 8R X 10C

	1 SCHEDULE TIME	2 STUDY1 PIV1A	3 STUDY1 PIV2A	4 STUDY1 PIV3A	5 -N-	6 -SUM-	7 -MEAN-	8 -VARIANCE-	9 -STDEV-	10 -CI95-
1	0.	10.	9.997536	10.	3.	29.99754	9.999179	3.814697×10^{-6}	.001953125	.00484884
2	5.	9.439186	9.444855	9.440242	3.	28.32428	9.441428	7.629395×10^{-6}	.002762136	.006857296
3	10.	8.908329	8.914221	8.911521	3.	26.73407	8.911357	8.583069×10^{-6}	.002929687	.00727326
4	15.	8.4098	8.407098	8.414143	3.	25.23104	8.410347	1.239777×10^{-5}	.003521046	.008741371
5	30.	7.080837	7.070375	7.068404	3.	21.21962	7.073206	4.482269×10^{-5}	.006694975	.01662099
6	60.	5.012142	5.018169	4.999849	3.	15.03016	5.010053	8.72612×10^{-5}	.00934137	.02319094
7	120.	2.501626	2.54935	2.33014	3.	7.381116	2.460372	.01328957	.1152804	.2861958
8	240.	.6419001	.6980882	.4280352	3.	1.768024	.5893412	.02030399	.1424921	.3537617

Figure 8. Results-table after using POOLTABS.

DELETETABS

by
Nick Holford
Division of Clinical Pharmacology
University of California, San Francisco
and
Harold Perry
Bolt Beranek and Newman Inc.

Summary

DELETETABS deletes a set of tables.

Calling Sequence

```
RAC PUBLIC DELETETABS <GO>
CALL DELETETABS <GO>
```

Citation

No citation necessary.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things to Remember

Be absolutely certain that the specified tables are expendable.

The public procedure DELETETABS deletes a set of tables. When you call DELETETABS, the procedure asks you for the name of a TEXT ARRAY containing the names of the tables you want to delete. It then displays a list of the specified tables and asks whether it is OK to delete them.

Be sure to exercise extreme caution when using this procedure.

EDCOL

by Karl Kuhlmann
Stanford Research Institute

Summary

EDCOL enables you to edit an entire column of a table (containing text or numbers) using the PROPHET text editor. EDCOL is particularly useful for editing large texts saved in table cells.

Calling Sequence

```
RAC PUBLIC EDCOL <GO>
CALL EDCOL(data-table, col-id)
           <table>    <fixed>
```

Citation

No citation necessary.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things to Remember

You must treat the row number indications [n.] carefully.

Consult the User's Manual for further description of the PROPHET text editor.

When you need to revise entries in a column of a PROPHET table, you can use the public procedure EDCOL to do this through one of PROPHET's two text editors. The command is

```
CALL EDCOL(data-table, col-id)
```

where data-table is the name of the table, and col-id is the FIXED number of the column containing the data to be edited. Note that the data type of the column must be TEXT or NUMBER.

Actually, it is probably easier to use the EDIT <table> command to revise numerical data. However, if your table contains text information it is probably easier to use EDCOL.

EDCOL converts all data in the column into a text to be edited. The row numbers are placed on a separate line in brackets [n.] between the contents of the cells. You must treat the row indications carefully because when you leave the editor, the procedure uses them to return the edited data to the table.

The editor presents empty cells as "<EMPTY>".

Consult the following example for further illustration.

Suppose MYSEQS is a saved table of several nucleotide sequences to which you have added an extra "DESCRIPTION" column (see Figure 1). To add the author of each sequence to the descriptions, you can use EDCOL (see Figure 2). You then use the PROPHET editor to add the authors of each sequence (see Figure 3). Note that you must preserve the line markers to guarantee that the revised information is correctly saved back in the original table. After you exit from the PROPHET editor, the revised data are automatically saved (see Figure 4).

MYSEQS 3R X 5C

0 NAME	1 TYPE	2 VALUE	3 MAKER	4 DATE	5 DESCRIPTION
1. PSTV	TEXT	CGGAACUAAACUCG-	WOOD	9/2/80	POTATO SPINDLE TUBER VIROID. -
2. TYRTRNA	TEXT	GCUUCCCGAUAAGG-	WOOD	9/2/80	E. COLI TYROSINE TRNA PRECURS-
3. IGG1	TEXT	TCGGGGACATGGGA-	WOOD	1/30/81	MOUSE IMMUNOGLOBULIN HEAVY CH-

Figure 1. MYSEQS data table.

```
>RAC PUBLIC EDCOL<
>CALL EDCOL(MYSEQS,5)<

[1.]
POTATO SPINDLE TUBER VIROID. 359 BASES.
[2.]
E. COLI TYROSINE TRNA PRECURSOR. 126 BASES.
[3.]
MOUSE IMMUNOGLOBULIN HEAVY CHAIN GAMMA1 CONSTANT REGION WITH 3 INTERV-
ENING SEQUENCES. 1823 BP.
```

Figure 2. Calling EDCOL and the initial format of the DESCRIPTION column.

```
[1.]
POTATO SPINDLE TUBER VIROID. 359 BASES.
(FROM GROSS, H.J.)
[2.]
E. COLI TYROSINE TRNA PRECURSOR. 126 BASES.
(FROM KHORANA, H.G.)
[3.]
MOUSE IMMUNOGLOBULIN HEAVY CHAIN GAMMA1 CONSTANT REGION WITH 3 INTERV-
ENING SEQUENCES. 1823 BP.
(FROM HONJO, T.)
```

Figure 3. Revised DESCRIPTION information.

```
>DISPLAY MYSEQS FOR COLUMN 5 ROWHEIGHT=5<

MYSEQS  3R X 5C
 0                      5
 NAME                   DESCRIPTION

 1. PSTV                POTATO SPINDLE TUBER VIROID. -
                        359 BASES.
                        (FROM GROSS, H.J.)


 2. TYRTRNA             E. COLI TYROSINE TRNA PRECURS-
                        OR. 126 BASES.
                        (FROM KHORANA, H.G.)


 3. IGG1                MOUSE IMMUNOGLOBULIN HEAVY CH-
                        AIN GAMMA1 CONSTANT REGION WI-
                        TH 3 INTERVENING SEQUENCES. 1-
                        823 BP.
                        (FROM HONJO, T.)
```

Figure 4. MYSEQS with revised DESCRIPTIONS.

TABLE2FILE

by David Fram
Bolt Beranek and Newman Inc.

Summary

TABLE2FILE writes a PROPHET table into a formatted ASCII file.

Calling Sequence

```
RAC PUBLIC TABLE2FILE <GO>
CALL TABLE2FILE(data-table)
                <table>
```

Entry

```
CALL TABLE2FILEQT(data-table, file-name, rowname-opt)
                  <table>     <text>     <bool>
```

Citation

No citation necessary.

Questions or Comments

Contact David Fram on the PROPHET line.

Things to Remember

Only tables containing columns of type NUMBER, FLOAT, FIXED, TEXT, DESCRIPTOR, or CHEMTEXT can be used with TABLE2FILE.

The public procedure TABLE2FILE enables you to write a PROPHET table into a formatted ASCII file. You can use any PROPHET table with columns of type TEXT, NUMBER, FLOAT, FIXED, DESCRIPTOR, or CHEMTEXT with TABLE2FILE.

Calling TABLE2FILE

To use TABLE2FILE, type the command:

```
CALL TABLE2FILE(data-table)
```

The procedure asks you to supply the name of the file to receive the data. The file should be of the form filename.ext. If you do not specify an extension, it is defaulted to .DAT. (See the <u>PROPHET</u> <u>User's</u> <u>Manual</u> for a description of filenames.)

If the table has row names, TABLE2FILE asks whether you want them written into the file.

Data File Organization

TABLE2FILE creates an ASCII data file with the organization shown in Table 1.

Table 1. Organization of the ASCII Data File

Card 1	NROWS NCOLS	NFMTCRDS (3I10)
Cards 2 to (NFMTCRDS+1)		FORTRAN format descriptor specifying how the data is written
Cards (NFMTCRDS+2) to (NFMTCRDS+1 + NCOLS		Column headings: 1 heading per card
Cards (NFMTCRDS+2 + NCOLS) onwards		The actual data from the table, written out row-wise, with each row written according to the format given on Card 2 *et. seq.* The first row in this section of the file is actually a "pseudo-row" showing what values will be used to represent missing data (EMPTY) in each of the columns.

TABLE2FILE writes the file as a series of card images; each line is restricted to 72 characters.

If columns of the table have user-specified formats, the procedure maintains these formats. Where formats are missing, the procedure tries to choose reasonable formats based on the data type of the column.

For each column, TABLE2FILE chooses a representation for EMPTY that depends on the column's type:

- a string of #'s for textual columns
- 999 for columns with an F format
- %MAXPOSFLOAT for columns with an E format

The procedure issues an error message if it encounters a nonempty datum that will encode to the value chosen to represent missing data.

Quiet
Entry
TABLE2FILEQT

There is also a quiet entry, TABLE2FILEQT. Use the command

```
CALL TABLE2FILEQT(data-table, file-name,
                          rowname-opt)
```

where file-name is the TEXT name of the file you want to receive the data, and rowname-opt is a BOOL that, if TRUE, causes the row names to be written into the file.

2. Graph
Procedures

2. Graph
Procedures

2. Graph Procedures

DTOGPOINT	finds graph coordinates of tablet-indicated points	2-1
EDITPOINTS	edits the points of a graph by using the graphics tablet.	2-3
DIGITIZE	creates a graph from input traced on the graphics tablet.	2-11
GRAPH3D	specifies data to be displayed in three dimensions by DRAW3D or MULT3D.	2-21
DRAW3D	displays and manipulates data in three dimensions	2-29
MULT3D	displays and manipulates several sets of data in three dimensions	2-43
BARGRAPH	constructs a bar graph from percentile data .	2-47
XFROMY	computes X values for indicated Y values of a graph (optionally accepts Y values of the form ED50)	2-53

DTOGPOINT

by David Fram
Bolt Beranek and Newman Inc.

Summary

DTOGPOINT finds the X and Y coordinates of a tablet-indicated point on a PROPHET graph.

Calling Sequence

```
RAC PUBLIC DTOGPOINT <GO>
CALL DTOGPOINT(graph-X, graph-Y)
               <float>  <float>
```

Entry

```
CALL DTOGPOINTQT(dis-X,  dis-Y, graph-X, graph-Y, graph-right-Y)
                 <float> <float> <float>  <float>     <float>
```

Citation

No citation necessary.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things to Remember

You must display a graph on the screen before you can indicate a point with the light pen.

You must declare graph-X and graph-Y before calling DTOGPOINT.

DTOGPOINT works only on display terminals with a graphics tablet.

DTOGPOINT determines the X and Y coordinates of a point you indicate on a graph with the light pen; the graph must be displayed on the screen.

Using DTOGPOINT

To use the procedure, type the command

```
CALL DTOGPOINT(graph-X, graph-Y)
```

where graph-X and graph-Y have been declared to be FLOAT variables.

A bell prompts you to indicate a point with the light pen. After you do this, graph-X and graph-Y contain the X and Y graph coordinates of the indicated point.

Quiet Entry, DTOGPOINTQT

DTOGPOINT also has a quiet entry, DTOGPOINTQT. Like DTOGPOINT, this entry converts display coordinates to graph coordinates. (The graph must be displayed on the terminal screen before you call DTOGPOINTQT.) However, DTOGPOINTQT does not require that you indicate the point with a light pen. Instead, you must supply the point's display coordinates as FLOAT arguments.

To use DTOGPOINTQT, type the command

```
CALL DTOGPOINTQT(dis-X, dis-Y, graph-X,
  graph-Y, graph-right-Y)
```

where all of the arguments are FLOAT variables. On return from the procedure, graph-X and graph-Y contain the graph coordinates of the point defined by dis-X and dis-Y. If your displayed graph also has a right Y axis, graph-right-Y contains the right Y-axis coordinate of the specified point.

Be sure that you declare all the arguments to be FLOAT variables before you call the the procedure.

EDITPOINTS

by
Harold Perry
Bolt Beranek and Newman Inc.
and
Karl Kuhlmann
Stanford Research Institute

Summary

EDITPOINTS enables you to use a graphics tablet to "edit" the points of a curve. Options include marking indicated points with specified symbols and deleting indicated points. EDITPOINTS is particularly useful for marking special points with new symbols and using FOR CLAUSES in PROPHET commands to process them appropriately.

Calling Sequence

```
RAC PUBLIC EDITPOINTS <GO>
CALL EDITPOINTS(graf)
                <graph>
```

Citation

Perry HM and Kuhlmann K. EDITPOINTS, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things to Remember

Data removed from your curve with the DELETE command are unrecoverable.

EDITPOINTS works only on a display terminal with a graphics tablet.

EDITPOINTS is a curve-editing tool with several features designed to help you identify and process interesting points of a curve. For example, you can use EDITPOINTS to mark outliers of a graph named "G" with a special symbol, say "*". Subsequently, you can use a FOR clause to exclude these points from a linear fit. The command

```
FIT LINE TO G FOR POINTS < > '*' <GO>
```

will automatically omit any point whose symbol is *.

You can also use EDITPOINTS to identify different groups of data based in your graph. You just identify different groups by using the MARK command and then apply a FOR clause to the POINTS table of your graph for selective processing. Consult the example at the end of this section for further elaboration.

EDITPOINTS also enables you to delete points from a curve simply by pointing to them.

Calling EDITPOINTS

To use the procedure, type

```
CALL EDITPOINTS(graf)
```

where graf is the name of the graph containing the curve(s) you wish to edit. EDITPOINTS displays your graph, along with a menu of light buttons.

If your graph contains only one curve, the procedure prompts you with a bell, signalling that it is ready for you to type or indicate a command.

If your graph contains two or more curves, the procedure asks you for the index of the first curve you would like to edit. Once you answer, a bell prompts you for a command. EDITPOINTS rings a bell whenever it is awaiting input from you.

EDITPOINTS Commands

The EDITPOINTS commands are described in detail below.

MARK

The MARK command lets you mark indicated points with new symbols. When you give this command, the procedure asks how you would like to choose symbols for marking points. The three MARK modes are R, S, and D.

Type R if you want to label each point with its row number from the points table of the graph, i.e., the index of each point.

Type S if you would like to mark every point you indicate with the same symbol; EDITPOINTS will ask you for the symbol.

Type D if you want to mark each point you indicate with a different symbol; the procedure will prompt you for a symbol before you mark each point.

After you give the MARK command and specify the mode and symbol (if appropriate), use the light pen to indicate the points whose symbols you wish to change. EDITPOINTS displays the new symbols over the old ones.

UNMARK	The UNMARK command restores the points you indicate to their default symbols. A point's default symbol is the symbol it had when the curve was first made.
SYMBOL	The SYMBOL command lets you change the MARK mode. When you give this command, the procedure asks how you would like to choose symbols for marking points. You may then continue marking points. (See description of "MARK" above.)
RADIUS	The RADIUS command enables you to change the radius of point recognition. When you call EDITPOINTS, the radius is set to 30 display units. The procedure asks you for a new radius value when you issue this command.
ZOOM	The ZOOM command magnifies a section of your graph to facilitate point editing. When you give this command, EDITPOINTS asks you to specify the upper left hand corner and lower right hand corner of a rectangle. This rectangular portion is then displayed so that it is expanded to fill the screen. This command is particularly useful for congested regions of your graph. You may issue successive ZOOM commands.
UNZOOM	The UNZOOM command restores the display of your graph to the way it was when you first called EDITPOINTS.
DELETE	The DELETE command enables you to delete indicated points from the curve. The data points are removed from the points table and are unrecoverable. This command operates in two modes. The procedure either deletes indicated points automatically, or asks you whether it is OK to delete indicated

	points before actually doing so. EDITPOINTS asks which mode you prefer.
CLEAN	The CLEAN command redisplays the graph with the new symbols you have assigned and without the points you have deleted.
HELP	The HELP command displays brief descriptions of the effects of the various EDITPOINTS commands.
EXIT	The EXIT commmand enables you to leave EDITPOINTS.
CURVE	The CURVE command enables you to choose which curve you want to edit on a multi-curve graph. The procedure asks for the curve index. This button is not displayed if your graph has only one curve.
Note	When you give a MARK, UNMARK, or DELETE command, tablet-indicated points are processed accordingly. At any time, you may indicate any of the other commands. EDITPOINTS continues to process tablet-indicated points the same way until you give a new MARK, UNMARK, or DELETE command.

Example

The following example illustrates the use of EDITPOINTS to achieve an improved linear fit. A line is fitted to data (See Figure 1), resulting in the fit results shown in Figures 2 and 3.

EDITPOINTS is then used to delete an extraneous data point and mark two outliers with new symbols (See Figures 4 to 6). A line is then fitted again to the data, but this time the outliers are omitted by the use of the FOR CLAUSE:

```
     FOR POINTS < > '*'
```

Consult Figures 7 and 8 for the fit results of the second linear fit.

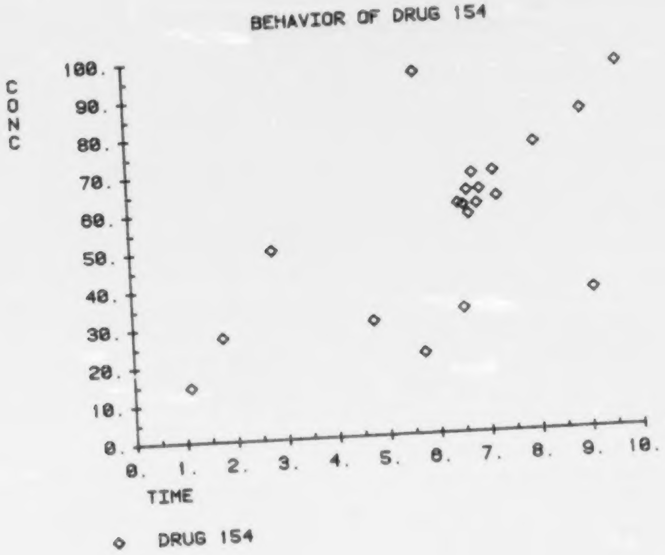


Figure 1. Original D154 graph of data.

```
>FIT LINE TO D154<
```

```
LINEAR FIT TO:
CURVE 1 OF D154
NUMBER OF DATA POINTS = 20
CORRELATION COEFFICIENT R = .6353244  R-SQUARED = .4036371
STANDARD DEVIATION OF REGRESSION = 18.332
```

PARAMETER TABLE

PARAMETER	FITTED VALUE	STANDARD DEVIATION	T-VALUE	SIG. LEV.
INTERCEPT	15.28884	12.47343	1.225713	.2361
SLOPE	6.523549	1.868992	3.49041	.0026

ANALYSIS OF VARIANCE TABLE

SOURCE	SUM OF SQUARES	D.F.	MEAN SQUARE	F VALUE	SIG. LEV.
REGRESSION	4094.232	1.	4094.232	12.18296	.00261
RESIDUAL	6049.117	18.	336.0621		

Figure 2. Results of "FIT LINE TO D154" command.

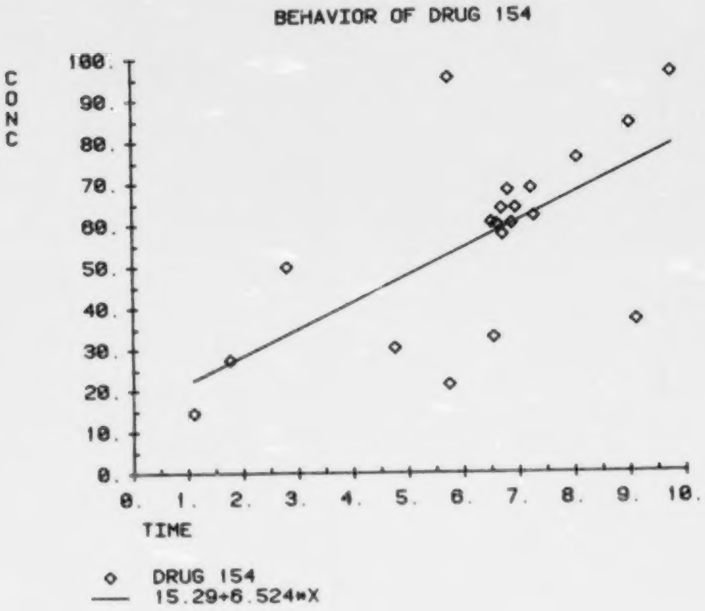
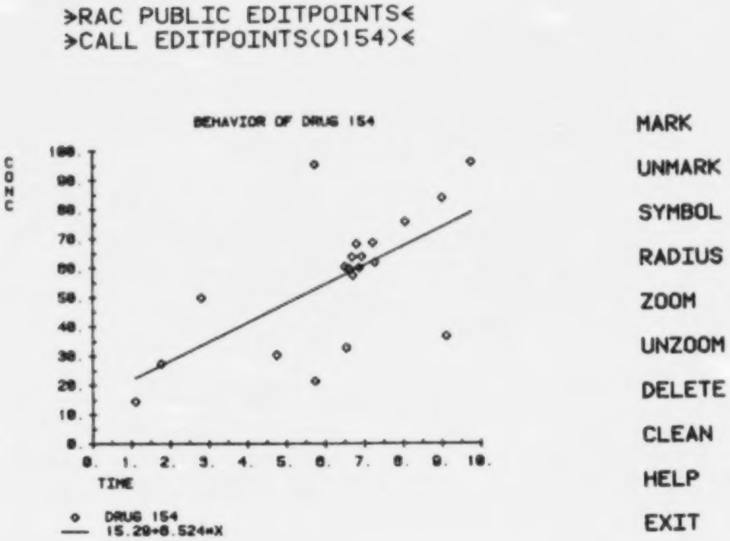


Figure 3. A Graph of the fit.



```
>RAC PUBLIC EDITPOINTS<
>CALL EDITPOINTS(D154)<
```

BEHAVIOR OF DRUG 154
CONC
100.
90.
80.
70.
60.
50.
40.
30.
20.
10.
0.
0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10.
TIME
◇ DRUG 154
—— 15.29+6.524*X
MARK
UNMARK
SYMBOL
RADIUS
ZOOM
UNZOOM
DELETE
CLEAN
HELP
EXIT

Figure 4. Calling EDITPOINTS and the initial quiz environment.

```
>HELP<
WHEN YOU HEAR THE BELL, TYPE OR INDICATE:
        "MARK"   TO MARK THE INDICATED POINTS WITH NEW SYMBOLS.

        "UNMARK" TO RESTORE THE SYMBOLS OF THE INDICATED POINTS BACK
                 TO THEIR DEFAULT SYMBOLS.

        "SYMBOL" TO CHANGE THE SYMBOL OR THE WAY SYMBOLS ARE SELECTED.
        (TYPE    R       TO MARK POINTS WITH ROW NUMBERS
                 S       TO MARK POINTS WITH THE SAME SYMBOL
                 D       TO MARK DIFFERENT POINTS WITH
                         DIFFERENT SYMBOLS).

        "RADIUS" TO CHANGE THE RADIUS (IN DISPLAY UNITS) OF POINT
                 RECOGNITION.

        "ZOOM"   TO ZOOM IN ON A SPECIFIED PORTION OF THE GRAPH.

        "UNZOOM" TO RESTORE A GRAPH TO ITS ORIGINAL DISPLAY.

        "DELETE" TO DELETE THE INDICATED POINTS FROM THE GRAPH.

        "CLEAN"  TO REDISPLAY THE GRAPH WITH THE NEW SYMBOLS.

        "HELP"   TO GET THIS MENU.

        "CURVE"  TO INDICATE WHICH CURVE TO EDIT POINTS FOR.

        "EXIT"   TO STOP MARKING POINTS.
```

Figure 5. EDITPOINTS' response to the HELP command.

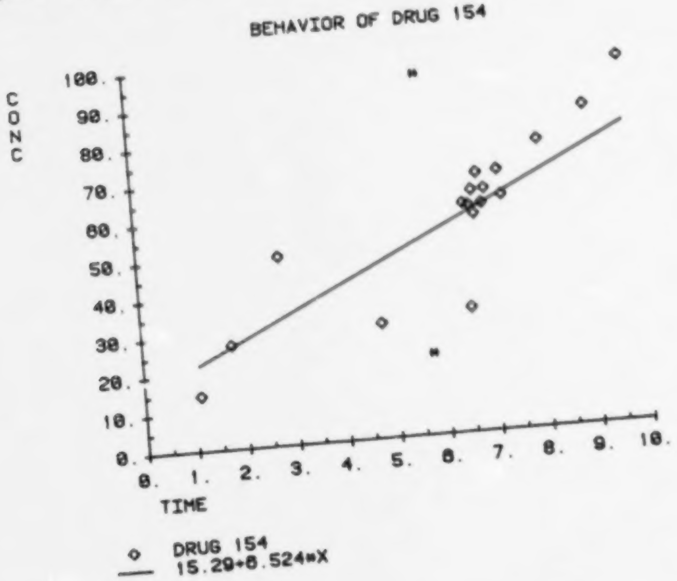


Figure 6. D154 after EDITPOINTS was used to DELETE an extraneous data point, and MARK 2 outliers with "*". (Compare with Figure 1.)

```
>FIT LINE TO D154 FOR POINTS <> '*'<

LINEAR FIT TO:
CURVE 1 OF D154 FOR POINTS < > *
NUMBER OF DATA POINTS = 17
CORRELATION COEFFICIENT R = .8767063  R-SQUARED = .7686139
STANDARD DEVIATION OF REGRESSION = 10.40184

PARAMETER TABLE
```

PARAMETER	FITTED VALUE	STANDARD DEVIATION	T-VALUE	SIG. LEV.
INTERCEPT	8.640346	7.344675	1.176409	.2578
SLOPE	7.84838	1.111857	7.058804	.0001

```
ANALYSIS OF VARIANCE TABLE
```

SOURCE	SUM OF SQUARES	D.F.	MEAN SQUARE	F VALUE	SIG. LEV.
REGRESSION	5391.17	1.	5391.17	49.82672	.0001
RESIDUAL	1622.976	15.	108.1984		

Figure 7. Results of "FIT LINE TO D154 FOR POINTS < > '*'" command. (Compare with Figure 2.)

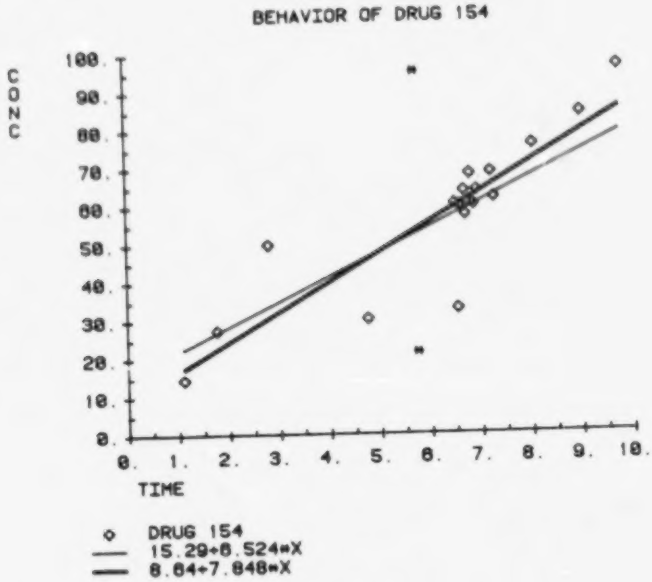


Figure 8. The final version of D154, showing the edited points and both linear fits.

DIGITIZE

by
Karl Kuhlmann
Stanford Research Institute
and
Harold Perry
Bolt Beranek and Newman Inc.

Summary

DIGITIZE enables you to enter X,Y data into a PROPHET graph or table by using the graphics tablet to indicate coordinates.

Calling Sequence

```
RAC PUBLIC DIGITIZE <GO>
CALL DIGITIZE <GO>
```

Entries

To save X,Y data in a graph:

```
CALL DIGITIZE <GO>

CALL DIGITIZEQK(graf,   axes-labels,   axes-limits,
               <text>  <text array>  <float array array>

                  curve-specs)
               <text array array>
```

To save X,Y data in a table:

```
CALL DIGITIZEPTS <GO>

CALL DIGITIZEPTSQK(table-name, axes-labels,     axes-limits)
                     <text>    <text array> <float array array>
```

Citation

Kuhlmann K and Perry HM. DIGITIZE, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things to Remember

DIGITIZE works only on a display terminal with a graphics tablet.

DIGITIZE enables you to save tablet-indicated coordinates in a PROPHET table or graph. To digitize a curve (or curves), simply place the curve(s) you want to trace on the tablet. Then invoke, DIGITIZE by typing the command:

```
CALL DIGITIZE <GO>
```

Graph Entry DIGITIZE

The procedure first asks whether you wish to use a common set of X values. If you do, answer YES and consult "Using a Common Set of X Values" below for further instructions.

DIGITIZE next asks for information to define the X and Y axes. For each axis, the procedure prompts you for:

- its label
- its minimum value (the value at the origin)
- its maximum value

The procedure then asks you to use the light pen to specify the origin, the end of the X axis, and the end of the Y axis on the graphics tablet. (When you hear the bell, DIGITIZE is waiting for a tablet indication.)

You are now ready to trace in your curve(s). (Refer to Figure 1 for an illustration of the interaction with DIGITIZE.) While entering the points of a curve, you may also type any of the following commands, which control the digitizing interaction:

DEL	deletes the last point entered in the current curve
NUM	indicates the number of points entered in the current curve
AXES	enables you to respecify axes Respecification of the axes includes rescaling, i.e., respecifying the minimum and maximum value of each axis and/or using the light pen to respecify the origin and ends of each axis.
REPEAT	lets you start digitizing the current curve again
NEXT	indicates that you have finished digitizing the current curve and want to digitize a new one on the same graph
EXIT	indicates that you have finished digitizing your last curve
HELP	lists discriptions of each command

When you finish digitizing a curve (i.e., after you issue an EXIT or a NEXT command), DIGITIZE asks you to specify:

- the name of the curve just completed
- the curve symbol
- whether or not the curve should be connected

Once you issue the EXIT command, the procedure displays a completed graph of the digitized curve(s). You can then choose a name for the graph. Figure 2 shows an example of a graph constructed with DIGITIZE.

```
>RAC PUBLIC DIGITIZE<
>CALL DIGITIZE<

ENTER LABEL FOR X-AXIS:TIME<
TYPE IN TIME VALUE AT ORIGIN OF X-AXIS:0<
TYPE IN TIME VALUE AT END OF X-AXIS:24<

ENTER LABEL FOR Y-AXIS:T
E
M
P<
TYPE IN T
E
M
P VALUE AT ORIGIN OF Y-AXIS:0<
TYPE IN T
E
M
P VALUE AT END OF Y-AXIS:90<

INDICATE ORIGIN ON THE TABLET

INDICATE END OF X-AXIS ON THE TABLET

INDICATE END OF Y-AXIS ON THE TABLET

DO YOU WISH TO USE A COMMON SET OF X-VALUES FOR ALL CURVES?YES<
AVAILABLE COMMANDS ARE:

DEL    NUM    NEXT    EXIT    REPEAT    AXES    HELP
EMPTY  NEWX
(WHEN YOU HEAR THE BELL YOU MAY ENTER A NEW POINT ON THE
TABLET. AT ANY TIME, YOU MAY TYPE ANY OF THE ABOVE COMMANDS).

TRACE IN X-VALUES

NEXT<
AVAILABLE COMMANDS ARE:

DEL    NUM    NEXT    EXIT    REPEAT    AXES    HELP
EMPTY  NEWX
TRACE IN CURVE 1

NEXT<
ENTER LABEL OF CURVE JUST FINISHED:SENSOR 1 READINGS<

ENTER SYMBOL OF CURVE JUST FINISHED:
%SQUARE<

SHOULD CURVE BE CONNECTED?YES<
AVAILABLE COMMANDS ARE:

DEL    NUM    NEXT    EXIT    REPEAT    AXES    HELP
EMPTY  NEWX
TRACE IN CURVE 2

EXIT<

ENTER LABEL OF CURVE JUST FINISHED:SENSOR 2 READINGS<

ENTER SYMBOL OF CURVE JUST FINISHED:
%TRIANGLE<

SHOULD CURVE BE CONNECTED?YES<
WHAT NAME DO YOU WANT FOR THE GRAPH ?TEMPS<
DIGITIZE COMPLETED.  TOOK 5.444 CPU SECS>
```

Figure 1. Interacting with DIGITIZE.

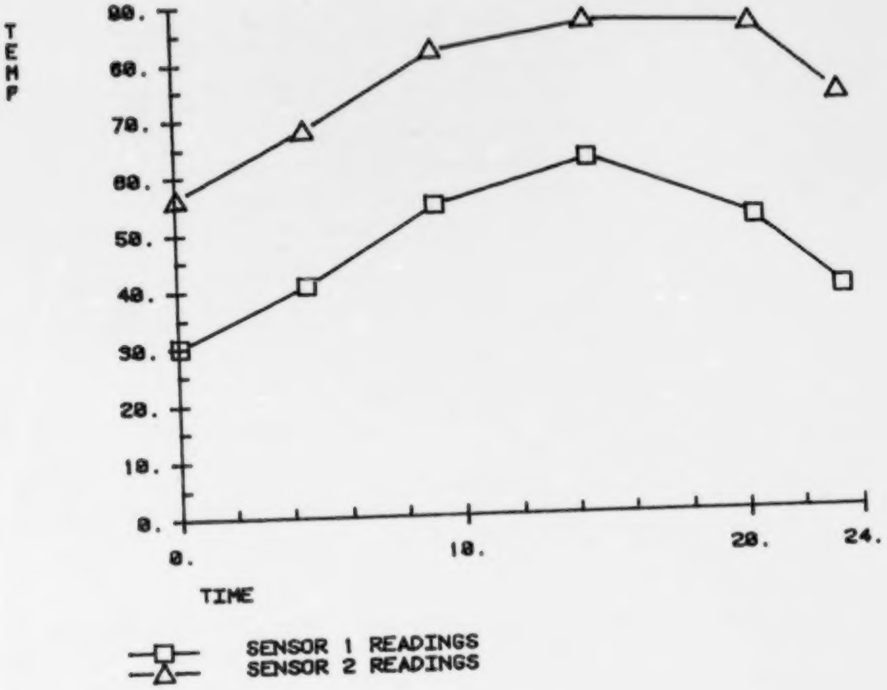


Figure 2. TEMPS graph constructed using DIGITIZE.

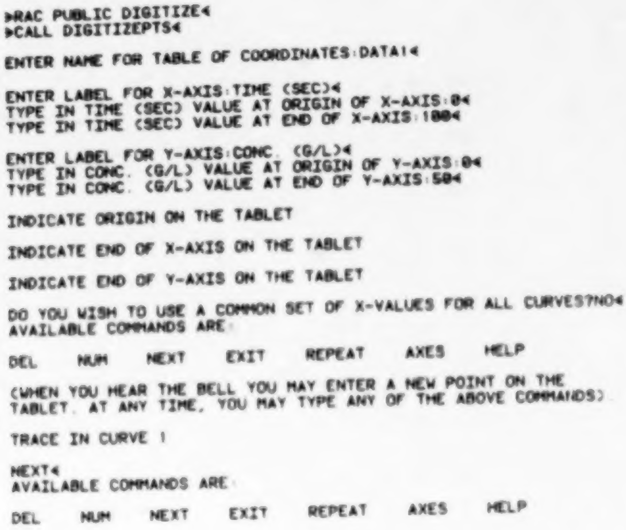
Table Entry
DIGITIZEPTS

To save the indicated coordinates in a table instead of in a graph, use the command:

```
CALL DIGITIZEPTS <GO>
```

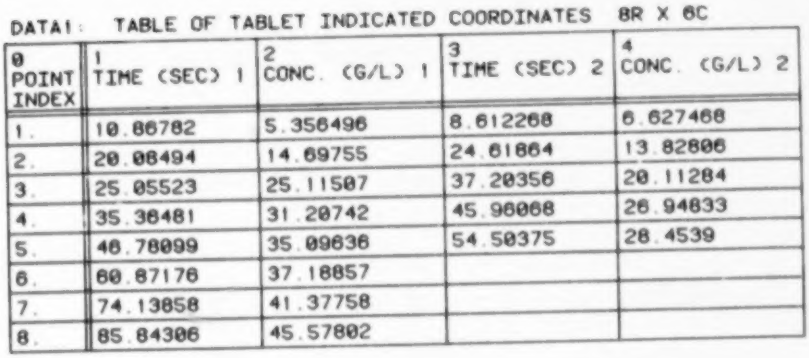
Once called, this entry behaves almost exactly as DIGITIZE does (see Figure 3). The only difference is that you must first indicate the name of a table for the digitized data. After you finish digitizing a curve with DIGITIZEPTS, you need not specify its name, its symbol, or whether it should be connected.

DIGITIZEPTS then creates a private table with the specified name and automatically stores the coordinates within it. The procedure uses the axes names as headings for the columns of this table. If you digitize several curves, it numbers the column headings accordingly (Figure 4).



```
▶RAC PUBLIC DIGITIZE◀
▶CALL DIGITIZEPTS◀

ENTER NAME FOR TABLE OF COORDINATES:DATA1◀

ENTER LABEL FOR X-AXIS:TIME (SEC)◀
TYPE IN TIME (SEC) VALUE AT ORIGIN OF X-AXIS:0◀
TYPE IN TIME (SEC) VALUE AT END OF X-AXIS:100◀

ENTER LABEL FOR Y-AXIS:CONC. (G/L)◀
TYPE IN CONC. (G/L) VALUE AT ORIGIN OF Y-AXIS:0◀
TYPE IN CONC. (G/L) VALUE AT END OF Y-AXIS:50◀

INDICATE ORIGIN ON THE TABLET

INDICATE END OF X-AXIS ON THE TABLET

INDICATE END OF Y-AXIS ON THE TABLET

DO YOU WISH TO USE A COMMON SET OF X-VALUES FOR ALL CURVES?NO◀
AVAILABLE COMMANDS ARE:

DEL     NUM     NEXT     EXIT     REPEAT     AXES     HELP

(WHEN YOU HEAR THE BELL YOU MAY ENTER A NEW POINT ON THE
TABLET. AT ANY TIME, YOU MAY TYPE ANY OF THE ABOVE COMMANDS)

TRACE IN CURVE 1

NEXT◀
AVAILABLE COMMANDS ARE:

DEL     NUM     NEXT     EXIT     REPEAT     AXES     HELP
```

Figure 3. Interacting with DIGITIZEPTS.



DATA1: TABLE OF TABLET INDICATED COORDINATES 8R X 6C

0 POINT INDEX	1 TIME (SEC) 1	2 CONC. (G/L) 1	3 TIME (SEC) 2	4 CONC. (G/L) 2
1.	10.86782	5.356496	8.612268	6.627468
2.	20.08494	14.69755	24.61864	13.82806
3.	25.05523	25.11507	37.20356	20.11284
4.	35.38481	31.20742	45.96068	26.94833
5.	46.78099	35.09636	54.50375	28.4539
6.	60.87176	37.18857		
7.	74.13858	41.37758		
8.	85.84306	45.57802		

Figure 4. DATA1 table constructed using DIGITIZEPTS.

Quick Graph Entry DIGITIZEQK

Quick versions of DIGITIZE and DIGITIZEPTS are also available. By including additional arguments in these entries, you can avoid preliminary questions concerning, for example, the names and limits of the axes. The quick version of DIGITIZE is DIGITIZEQK, which requires the arguments shown in Table 1.

Table 1. DIGITIZEQK Arguments.

Argument	Type	Description
graf	<text>	The name of the graph that will contain the digitized data
axes-labels	<text array>	The names of the X and Y axes, e.g., ['TIME', 'CONC']
axes-limits	<float array array>	The limits of the X and Y axes, e.g., [[0.,100.], [0.,1.]]
curve-specs	<text array array>	The specifications for each curve you wish to digitize. There should be a row for each curve such that the first entry is the curve name, the second is the curve symbol, and the third is either 'YES' or 'NO' depending on whether the curve should be connected, e.g., [['GROUP 1', '+', 'YES'], ['GROUP 2', '0', 'YES']].

When you call DIGITIZEQK, the procedure immediately asks you to specify the beginning and end of each axis and then to digitize your data.

When you specify NEXT, you are asked to digitize the data in the next curve. DIGITIZEQK expects you to digitize a number of curves equal to the number of rows in the curve-specs argument. If you need to digitize extra curves, the procedure asks you for their names, their symbols, and whether they should be connected.

When you EXIT, DIGITIZEQK creates a graph with the name you specified in the initial call.

```
>DCL ALBLS TEXT ARRAY<
>DCL ALMTS FLOAT ARRAY ARRAY<
>DCL SPECS TEXT ARRAY ARRAY<
>ALBLS = ['TIME (SEC)', 'CONCENTRATION (G/L)']<
>ALMTS[1] = [0., 100.]<
>ALMTS[2] = [0.,50.]<
>SPECS[1] = ['TEST 1', '*', 'YES']<
>SPECS[2] = ['TEST 2', '$', 'YES']<

>RAC PUBLIC DIGITIZE<
>CALL DIGITIZEQK('DATA1', ALBLS, ALMTS, SPECS)<
```

Figure 5. Using DIGITIZEQK.

Quick Table Entry DIGITIZEPTSQK

The quick version of DIGITIZEPTS is DIGITIZEPTSQK. It requires the arguments shown in Table 2.

Table 2. DIGITIZEPTSQK Arguments.

Argument	Type	Description
table-name	<text>	The name of the private table that will contain the digitized data
axes-labels	<text array>	The names of the X and Y axes. (These names will be used for column headings.)
axes-limits	<float array array>	The limits of the X and Y axes

Notice that the last two arguments are exactly analogous to the corresponding arguments of DIGITIZEQK. When you call this entry, it immediately asks you to use the tablet to digitize your data.

Using a Common Set of X Values

Sometimes you may need to digitize a group of curves for a given set of X values. When you call any of the DIGITIZE entries, the procedure asks whether you wish to do this. If you answer YES

(Y), you are first asked to specify the X values. (All entries display only the X axis when you are indicating these values.)

When you finish specifying the X values, type NEXT to digitize your first curve. Each entry proceeds by prompting you for the corresponding Y value for each of the X values you initially indicated. You must indicate the Y values in the same order that you specified the X values. Before each indication, the procedure draws a vertical line at the X value for which a tablet-indicated Y value is expected.

When you use any entry under this mode, two extra commands are available:

EMPTY	indicates that the next X value has an empty Y value
NEWX	enables you to specify a new X value while digitizing a curve

Note that you can specify Y values only for X values that you initially indicated or that you subsequently specify with the NEWX command.

GRAPH3D

by Eric J. Meyer
Bolt Beranek and Newman Inc.

Summary

GRAPH3D is used to specify data to be displayed three dimensionally by the procedure DRAW3D. It takes data out of a table or an array containing the data, or it asks you for an algebraic definition of the data. GRAPH3D stores the data in a PROPHET matrix (FLOAT ARRAY ARRAY) and lets you get a top-level copy of the result. Finally, it calls the procedure DRAW3D and displays the three-dimensional surface defined in GRAPH3D.

Calling Sequence

```
RAC PUBLIC GRAPH3D <GO>
CALL GRAPH3D <GO>
```

Citation

Meyer EJ. GRAPH3D, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Contact Eric Meyer on the PROPHET line.

Things to Remember

If you use a data table or an array, it must be in the format described below.

If you use GRAPH3D to get a top-level copy of the display matrix, remember to save the matrix before you log out, or it will be lost. It is only a FLOAT ARRAY ARRAY, not a TABLE.

GRAPH3D works on all types of terminals, but you can use DRAW3D only on a display terminal. Therefore, if you call GRAPH3D on a nondisplay terminal, it assumes you simply want to create and save the display matrix, and it does not attempt to call DRAW3D.

The purpose of GRAPH3D is to create a PROPHET matrix (FLOAT ARRAY ARRAY) representing a rectangular grid of samples from a surface in three-dimensional space. Each row of this matrix represents one sample point on the surface, with columns 1, 2, and 3 being the point's X, Y, and Z coordinates, respectively.

Think of X and Y as independent variables, with Z the dependent variable. Thus, by running GRAPH3D you are defining some function F where Z = F(X,Y). X and Y are sampled at discrete points in some rectangular region of 3-space, and Z is the height above the XY plane at each sample point.

Specifying (X,Y,Z)

GRAPH3D operates by asking a series of questions to define first the X data, then the Y data, and finally the Z data. Although X, Y, and Z can each come from a variety of sources, only some of the possible combinations are logically meaningful. This means that the available options for Y and Z depend on your previous answers to the questions about X. The possible forms of input are:

- X from ROW NUMBERS, Y from COLUMN NUMBERS, and Z from CELL(X,Y).

  The X values range from 1 to the number of rows of a table or an array. Similiarly, the Y values range from 1 to the number of columns of a table or an array. The Z value for each (X,Y) point is simply whatever number is at row X, column Y.

  Since GRAPH3D uses the entire table or array in creating the three-dimensional display, all columns of the table must be numeric. If any cells are empty, GRAPH3D prints a warning message, ignores the empty values, and continues.

- X, Y, and Z each from a COLUMN of a table or an array.

  The three columns must all be numeric, and the X and Y columns must not have empty cells. If any cell in the Z column is empty, GRAPH3D prints a warning message, ignores the point, and continues.

In general, every row of the table or array corresponds to one point in 3-space. The only exception to this is when one row has exactly the same X and Y values as another. In this case, GRAPH3D takes the Z value for that X and Y from the row it finds last, and does not warn you about it.

- X and Y from a COLUMN, and Z from a FUNCTION.

  In this case, exactly the same restrictions apply for the X and Y data: the relevant columns must be numeric and completely filled. The difference is that Z is defined as an algebraic function of X and Y. Methods of specifying a function for Z are described below.

- X from a RANGE, Y from a COLUMN, and Z from a FUNCTION.

  To specify a range, you must supply the low and high values along the range, and the number of sample points to take between those two values.

  Thus, if you wished to specify X values of -2, -1, 0, 1, 2, and 3 using a RANGE, you would specify a low value of -2, a high value of 3, and 6 samples. Each of the X values is paired with each of the numbers in the Y data column to define a point, where Z is the value of the function at each X and Y.

  So, if you ask for 10 samples along the X axis, and the Y column contains 8 numbers, there will be a total of 10 x 8 = 80 points in the resulting surface to be displayed.

- X and Y from a RANGE, and Z from CELLS of a table or array.

  This is similar to the first form of input (X from ROW NUMBERS, Y from COLUMN NUMBERS) except that X and Y can have some more meaningful range than just 1 to the number of rows or columns. The X and Y ranges are independent of each other. The i'th X value is paired with the j'th Y value, and the Z value comes from table or array cell(i,j).

When specifying ranges, be careful not to specify a larger number of samples than the data table contains Z data for. If, for example, your table has 10 rows, you should specify no more than 10 samples along X.

- X and Y data from a RANGE, and Z from a FUNCTION.

  In this case alone, you need not refer to a data table or an array -- GRAPH3D can generate all the numbers needed. Just as in the previous form of input, you specify low values, high values, and number of samples for X and Y independently. The Z values are the result of evaluating the function at each possible (X,Y) pair.

Specifying a Function for Z

When you specify that the Z values are to come from a function, GRAPH3D asks whether you have a function table. If you do not, simply type in an algebraic expression involving X and Y.

If you do have a function table, enter its name. The row names of the table should correspond to function names, and the table should contain a TEXT column of function equations. If it has more than one row, GRAPH3D asks you which function (which row name) you want to use. If the table has more than one column, the procedure asks which column contains the function equation.

GRAPH3D then asks whether you have a parameter table. If you do, enter its name and then if necessary, the column containing parameter values. The procedure assumes that the row names of the parameter table are the parameter names.

If you do not have a parameter table, GRAPH3D asks you to enter the parameter names and values, one by one. You should specify the name and value of each parameter that appears in the equation of the function.

Notes

If you refer to any table column for data, that column must be of type NUMBER, FLOAT, or FIXED. If your data come from an array, the array must be a FLOAT ARRAY ARRAY (matrix). Also, there can be no empty cells in a table or array column you reference.

If you define Z to be an algebraic function of X and Y, be careful that the function is well defined at all (X,Y) points. For example, the function Z = SQRT(X+Y) will be ill defined if at any point X+Y < 0. In such a situation GRAPH3D prints a warning message and ignores the problematic points.

Examples

Figure 1 illustrates how to graph all the data in a table or matrix. All columns of the table must be numeric, and all cells must be non-empty.

```
>CALL GRAPH3D<
X DATA FROM ROW NUMBERS, COLUMN, OR RANGE?ROWS<
NAME OF DATA TABLE OR ARRAY?MYDATA<
X FROM ROW NUMBERS IMPLIES Y FROM COLUMN NUMBERS
AND Z FROM CELL(X,Y)
DO YOU WANT A TOP-LEVEL COPY OF THE DISPLAY MATRIX?YES<
NAME =?MYDISMAT<
DO YOU WANT TO GO ON TO DRAW3D?YES<

GOING TO DRAW3D.
ARE YOU ON A LARGE-SCREEN (4014) TERMINAL?YES<
```

Figure 1. X from rows, Y from columns, Z from cells.

Figure 2 shows how you would graph X, Y, and Z data each from one column of a table or matrix.

```
>CALL GRAPH3D<
X DATA FROM ROW NUMBERS, COLUMN, OR RANGE?COLUMN<
NAME OF DATA TABLE OR ARRAY?MYDATA<
COLUMN NUMBER FOR X DATA?1<
X FROM COLUMN IMPLIES Y FROM COLUMN
COLUMN NUMBER FOR Y DATA?2<
Z DATA FROM COLUMN OR FUNCTION?COLUMN<
COLUMN NUMBER FOR Z DATA?3<
DO YOU WANT A TOP-LEVEL COPY OF THE DISPLAY MATRIX?YES<
NAME =?MYDISMAT<
DO YOU WANT TO GO ON TO DRAW3D?YES<

GOING TO DRAW3D.
ARE YOU ON A LARGE-SCREEN (4014) TERMINAL?YES<
```

Figure 2. X,Y,Z from columns.

To graph X and Y data from columns of a table or matrix, and Z from an algebraic function of X and Y, use GRAPH3D as shown in Figure 3.

```
>CALL GRAPH3D<
X DATA FROM ROW NUMBERS, COLUMN, OR RANGE?COLUMN<
NAME OF DATA TABLE OR ARRAY?MYDATA<
COLUMN NUMBER FOR X DATA?1<
X FROM COLUMN IMPLIES Y FROM COLUMN
COLUMN NUMBER FOR Y DATA?2<
Z DATA FROM COLUMN OR FUNCTION?FUNCTION<
DO YOU HAVE A FUNCTION TABLE?NO<
ENTER FUNCTION:  F(X,Y) =?SQRT(X*X + Y*Y)<
DO YOU WANT A TOP-LEVEL COPY OF THE DISPLAY MATRIX?YES<
NAME =?MYDISMAT<
DO YOU WANT TO GO ON TO DRAW3D?YES<

GOING TO DRAW3D.
ARE YOU ON A LARGE-SCREEN (4014) TERMINAL?YES<
```

Figure 3. X,Y from columns, Z from function.

Figure 4 shows how to graph X values from regular samples along a range, Y values from a column of a table or matrix, and Z values from an algebraic function.

```
>CALL GRAPH3D<
X DATA FROM ROW NUMBERS, COLUMN, OR RANGE?RANGE<
WHAT IS THE LOW VALUE ON THE X AXIS?-1<
WHAT IS THE HIGH VALUE?1<
HOW MANY SAMPLES ALONG X?10<
Y DATA FROM COLUMN OR RANGE?COLUMN<
NAME OF DATA TABLE OR ARRAY?MYDATA<
COLUMN NUMBER FOR Y DATA?1<
X FROM RANGE, Y FROM COLUMN
IMPLIES Z FROM FUNCTION
DO YOU HAVE A FUNCTION TABLE?YES<
NAME OF THE FUNCTION TABLE?FUNTAB<
NAME OF FUNCTION TO BE GRAPHED?EQ1<
COLUMN CONTAINING EQUATION OF FUNCTION EQ1?1<
DO YOU HAVE A PARAMETER TABLE?YES<
NAME OF THE PARAMETER TABLE?PARTAB<
COLUMN CONTAINING PARAMETER VALUES?1< [COMPILING...DONE]
DO YOU WANT A TOP-LEVEL COPY OF THE DISPLAY MATRIX?YES<
NAME =?MYDISMAT<
DO YOU WANT TO GO ON TO DRAW3D?YES<

GOING TO DRAW3D.
ARE YOU ON A LARGE-SCREEN (4014) TERMINAL?YES<
```

Figure 4. X from range, Y from column, Z from function.

Figure 5 illustrates how to graph X and Y values from samples along ranges, and Z values from cells of a table or matrix.

```
>CALL GRAPH3D<
X DATA FROM ROW NUMBERS, COLUMN, OR RANGE?RANGE<
WHAT IS THE LOW VALUE ON THE X AXIS?10<
WHAT IS THE HIGH VALUE?20<
HOW MANY SAMPLES ALONG X?11<
Y DATA FROM COLUMN OR RANGE?RANGE<
WHAT IS THE LOW VALUE ON THE Y AXIS?-3<
WHAT IS THE HIGH VALUE?3<
HOW MANY SAMPLES ALONG Y?7<
Z DATA FROM CELL(X,Y) OR FUNCTION?CELL<
NAME OF DATA TABLE OR ARRAY?MYDATA<
DO YOU WANT A TOP-LEVEL COPY OF THE DISPLAY MATRIX?YES<
NAME =?MYDISMAT< DO YOU WANT TO GO ON TO DRAW3D?YES<

GOING TO DRAW3D.
ARE YOU ON A LARGE-SCREEN (4014) TERMINAL?YES<
```

Figure 5. X,Y from ranges, Z from cells.

Figure 6 shows how you would graph X and Y values from samples along ranges, and Z values from an algebraic function.

```
>CALL GRAPH3D<
X DATA FROM ROW NUMBERS, COLUMN, OR RANGE?RANGE<
WHAT IS THE LOW VALUE ON THE X AXIS?-%PI<
WHAT IS THE HIGH VALUE?%PI<
HOW MANY SAMPLES ALONG X?16<
Y DATA FROM COLUMN OR RANGE?RANGE<
WHAT IS THE LOW VALUE ON THE Y AXIS?0.1<
WHAT IS THE HIGH VALUE?10<
HOW MANY SAMPLES ALONG Y?10<
Z DATA FROM CELL(X,Y) OR FUNCTION?FUNCTION<
DO YOU HAVE A FUNCTION TABLE?YES<
NAME OF THE FUNCTION TABLE?FUNTAB<
NAME OF FUNCTION TO BE GRAPHED?EQ1<
COLUMN CONTAINING EQUATION OF FUNCTION EQ1?1<
DO YOU HAVE A PARAMETER TABLE?NO<

ENTER PARAMETERS ONE BY ONE.
HIT <GO> TO EXIT.

NAME OF PARAMETER 1?A<
VALUE OF PARAMETER A?6<
NAME OF PARAMETER 2?B<
VALUE OF PARAMETER B?12<
NAME OF PARAMETER 3?< [COMPILING...DONE]
DO YOU WANT A TOP-LEVEL COPY OF THE DISPLAY MATRIX?YES<
NAME =?MYDISMAT<
DO YOU WANT TO GO ON TO DRAW3D?YES<

GOING TO DRAW3D.
ARE YOU ON A LARGE-SCREEN (4014) TERMINAL?YES<
```

Figure 6. X,Y from ranges, Z from function.

DRAW3D

by Eric J. Meyer
Bolt Beranek and Newman, Inc.

Summary

DRAW3D displays data three dimensionally. The data consist of a matrix (FLOAT ARRAY ARRAY) of three columns and many rows, where each row represents one (X,Y,Z) point. DRAW3D enables you to modify the appearance and orientation of the display interactively.

Calling Sequence

```
RAC PUBLIC DRAW3D <GO>
CALL DRAW3D(display-matrix)
            <float array array>
```

Entries

```
CALL REDRAW3D(display-matrix,         surface-xfmer,
              <float array array>     <float array array>

              screen-xfmer,           option-array)
              <float array array>     <bool array>

CALL D3DQT(   display-matrix,         surface-xfmer,
              <float array array>     <float array array>

              screen-xfmer,           option-array,
              <float array array>     <bool array>

              command)
               <text>
```

Citation

Meyer, EJ. DRAW3D, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Contact Eric Meyer on the PROPHET line.

Things to Remember

Whenever you are unsure of what to do next, the HELP command supplies the meanings of all the DRAW3D commands.

When using REDRAW3D and D3DQT, be sure to supply the surface and screen transformers in the correct order.

DRAW3D works only on display terminals.

DRAW3D graphically displays data from a matrix (FLOAT ARRAY ARRAY) such as those created by the procedure GRAPH3D. Basically, DRAW3D regards each row of the display matrix as one (X,Y,Z) point, and visits the points in the exact order they appear in the matrix, in the manner of a dot-to-dot drawing. When first displayed, the data are oriented so that you are looking toward the origin from a position in the positive XYZ octant, and they are scaled so that they fit into a cubical region of space whose front face corresponds to the physical screen of the terminal.

When you call DRAW3D, the procedure calculates the default position and orientation. Then it displays the data, the X, Y, and Z axes, and a list of commands on the left side of the screen.

You interact with DRAW3D by typing in one of the commands, followed by <GO>. The meanings of the various commands are explained below. In general, after you type a command, DRAW3D asks you for any additional information required to execute the command. Many of the commands require no more information and are executed immediately.

DISPLAY, for example, redraws the display to reflect whatever changes you have made since the last DISPLAY command. Other commands ask one or two questions before they execute. TURN, for example, asks you which axis to turn around, and how much to turn.

The OPTIONS command deserves special mention. There are several display options in DRAW3D, each of which is either ON or OFF. When you invoke the OPTIONS command, DRAW3D lets you change one of them. It asks which option you wish to change, and then types a message to tell you whether that option is now ON or OFF. The various options are summarized below under "DRAW3D Options".

DRAW3D Coordinate Systems

DRAW3D maintains two independent coordinate systems: the surface coordinate system, with axes X, Y, and Z, and the screen coordinate system, with axes U, V, and W. It is important to understand the difference between them.

The surface coordinate system (XYZ) is the system whose axes are displayed on the screen. The orientation of these axes is changeable: executing a TURN, MOVE, or SCALE command modifies them. For example, if you TURN around X by 90 degrees, the X axis is unchanged but Y and Z are rotated. If you TURN around a screen axis, for example U, all three surface axes are rotated.

The screen coordinate system (UVW) remains constant under all transformations you perform. The U axis points to the right, V points straight up, and W points out of the screen into your chest. These orientations do not change.

For example, if you TURN around U, the display turns through the desired angle, but V and W then remain pointing up and out, even though X, Y, and Z have turned.

The units of measurement in screen coordinates are also constant. The screen extends from -5.12 to 5.12 in U from left to right, from -3.9 to 3.9 in V bottom to top, and from -infinity to 7 in W back to front. The initial, default display fits into a cube in screen space centered at the origin and having edges 5 screen units long.

The plane W=7 is important in that it corresponds to the plane of the terminal screen itself. DRAW3D regards any point with a W coordinate greater than 7 as being "out of the screen" and thus invisible. Therefore, if you move a display toward you along the W axis too much, it may become truncated as the nearest portion disappears out of the screen.

DRAW3D Commands

The following is a list of DRAW3D commands. You can type any command in full, or use an abbreviation (e.g., its first letter). Of all the commands, only DISPLAY uses an appreciable amount of computer time. All the others execute very quickly.

The image on the screen is not redrawn until you give a DISPLAY command or a CLEAN command.

DISPLAY (D) Command

The DISPLAY command erases the screen, computes the appearance of the display given all the changes you have made since the last DISPLAY command, draws the display on the screen, and prompts you for the next command.

TURN (T) Command

The TURN command first asks which axis to turn around, and then how much to turn. You can turn around X, Y, or Z (the surface coordinate system axes) or around U, V, or W (the screen coordinate system axes).

TURN measures angles in degrees, not radians. Rotation is always right handed: if you imagine grasping the axis with your right hand, with your thumb pointing toward the high values on the axis, rotation by a positive angle follows your other four fingers.

SCALE (S) Command

The SCALE command asks which axis to scale along, and then by what factor. You can scale along X, Y, Z, U, V, or W. You can also scale along U, V, and W simultaneously by typing CTRL-E in response to the axis question.

The scale factor can be any real number. Scaling by 1 does nothing, scaling by 0 collapses the data along the specified axis, scaling by -1 reflects the axis through the origin, and so forth.

MOVE (M) Command

The MOVE command asks which axis to move along, and how much. You can move along X, Y, Z, U, V, or W. The units of movement depend on the coordinate system chosen: X, Y, and Z move in units that depend on the magnitude of your data. U, V, and W move in screen space, of which the visible region is from -5.12 to 5.12 along U, -3.9 to 3.9 along V, and -infinity to 7 along W.

When moving along the W axis, bear in mind that the display will be truncated along the W=7 plane, with the portion closer to you than that not shown because it is "in front of" the terminal screen.

	Moving along W will probably have no effect when the PERSPECTIVE option (described below) is OFF. The translation is done, but you cannot see it because of the parallel projection.
RESET (R) Command	The RESET command cancels the effect of all changes you have made since the beginning of the program, and returns you to the original, default display. This command is useful if you get confused and can no longer understand what you are seeing.
XFMSAVE (X) Command	The XFMSAVE command is discussed below in the section on advanced DRAW3D.
OPTIONS (O) Command	The OPTIONS command asks which of the DRAW3D display options you wish to change. When you respond, that option is changed and DRAW3D tells you the new state (ON or OFF) of the option. The options are described below under "DRAW3D Options".
FOCUS (F) Command	The FOCUS command modifies the intensity of perspective when the PERSPECTIVE option (described below) is ON. When PERSPECTIVE is OFF, you cannot use the FOCUS command. In a PERSPECTIVE display, the portion of the image that is "far away" is small, and the portion that is "near" is larger. At some point along the W axis (the axis perpendicular to the screen), the image is the same size as it would be in a parallel display. This is the focal plane. The FOCUS command enables you to move the focal plane forward or backward. It asks you for the number of units to move the focal plane along the W axis. A positive number moves the plane forward and diminishes the perspective. A negative number moves the plane backward and intensifies it.
HELP (H) Command	The HELP command gives you a description of each command and option while you are using DRAW3D, so that you do not need to have this documentation with you. Use HELP whenever you get confused or cannot remember how to do the next thing you want to do.

CLEAN (C) Command

The CLEAN command is exactly like the top-level PROPHET CLEAN command: it redraws the screen rapidly (i.e., faster than DISPLAY would because it does not have to recompute the display). This command is useful when the display has been garbled by line noise.

PLOT (P) Command

The PLOT command creates a plot file of the display as it currently appears on the screen. (The command text on the left side of the screen is not included.) Later, a paper plot of the display is sent to you.

EXIT (E) Command

The EXIT command terminates DRAW3D and returns you to top-level PROPHET.

DRAW3D Options

You can type any DRAW3D display option in full or you can abbreviate it by typing only its first letter.

In the list below, each option is described in the following form:

- Restrictions: any restrictions on using this option.
- Default: whether the option starts out ON or OFF
- ON: the effect the option has when ON
- OFF: the effect the option has when OFF

The options are:

GRID (G)

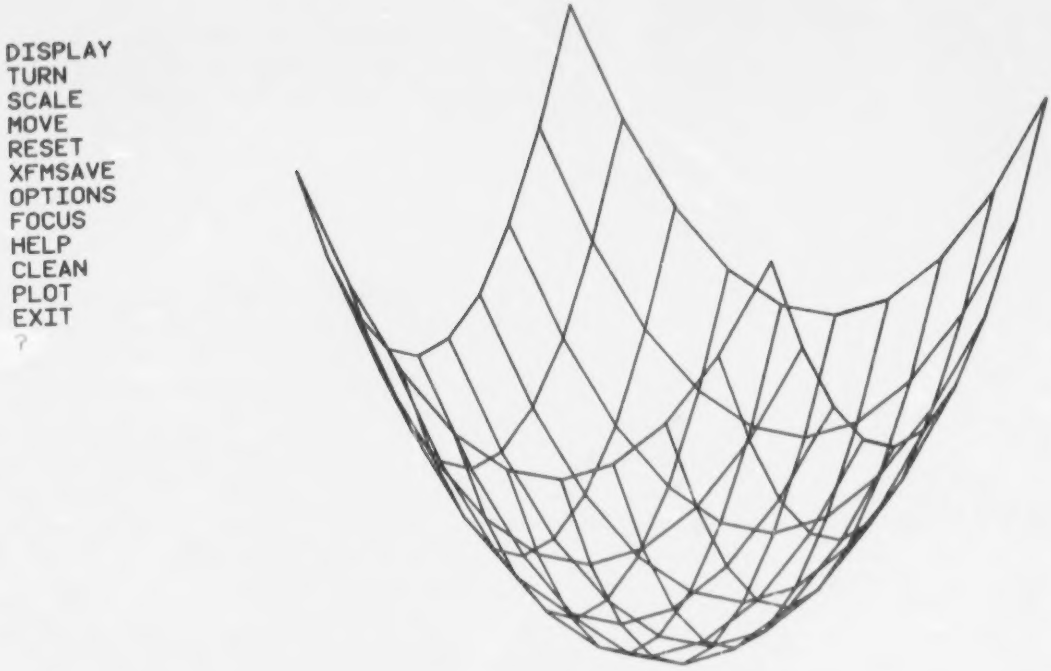
Restrictions: The NEEDLES option (described below) must be OFF to change GRID.

Default: ON

ON: DRAW3D connects each point on the surface to the next with a line, creating a display with the appearance of a sheet of cheesecloth.

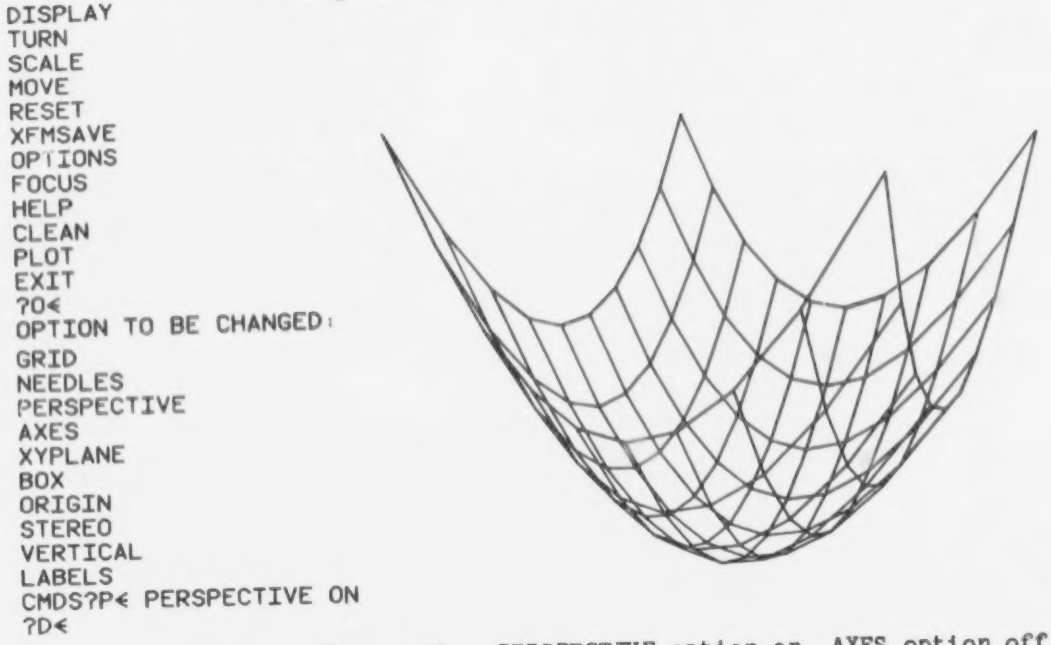
OFF: Alternate points on the surface are connected to the next point with a line, drawing exactly half the lines that would appear if GRID were ON. The surface appears to be covered with small rectangles

	like the spots on a leopard. If the surface is very bumpy, the rectangles are noticeably nonplanar (i.e., not rectangles at all, but 4-sided closed curves in three-dimensional space).
NEEDLES (N)	Restrictions: None Default: OFF ON: Instead of connecting one point to the next, the procedure connects each point to the XY plane, so that the display looks like a forest of needles coming up (or down, if negative) from the XY plane. OFF: When NEEDLES is off, the connectedness of the surface is defined by the state of the GRID option.
PERSPECTIVE (P)	Restrictions: None Default: OFF ON: The procedure scales each point in the display along the U and V axes according to its distance from the screen (which is related to the W coordinate of the point). (See the description of the FOCUS command.) If a line is "behind" the focal plane, it is shrunk along U and V. If it is between the screen and the focal plane, it is enlarged along U and V. If it is "in front of" the screen, it is not shown. OFF: There is no distance effect in the display. Points far away are just as far apart as corresponding points close up.
AXES (A)	Restrictions: None Default: ON ON: The procedure displays the X, Y, and Z axes and identifies them with the appropriate letters. The length of each axis' arrow is determined by the maximum data value along that axis. Note that the axes may be invisible even when the AXES option is ON, if you are looking at a region of space that does not cross them. OFF: Axes and their labels are omitted.



```
DISPLAY
TURN
SCALE
MOVE
RESET
XFMSAVE
OPTIONS
FOCUS
HELP
CLEAN
PLOT
EXIT
?
```

Figure 1a. PERSPECTIVE and AXES options off.



```
DISPLAY
TURN
SCALE
MOVE
RESET
XFMSAVE
OPTIONS
FOCUS
HELP
CLEAN
PLOT
EXIT
?O<
OPTION TO BE CHANGED:
GRID
NEEDLES
PERSPECTIVE
AXES
XYPLANE
BOX
ORIGIN
STEREO
VERTICAL
LABELS
CMDS?P< PERSPECTIVE ON
?D<
```

Figure 1b. PERSPECTIVE option on, AXES option off.

XYPLANE (X)

Restrictions: None

Default: OFF

ON: DRAW3D indicates the position of the XY plane by a 5 by 5 grid of lines, dividing the X and Y ranges into quarters.

OFF: The grid on the XY plane is not shown.

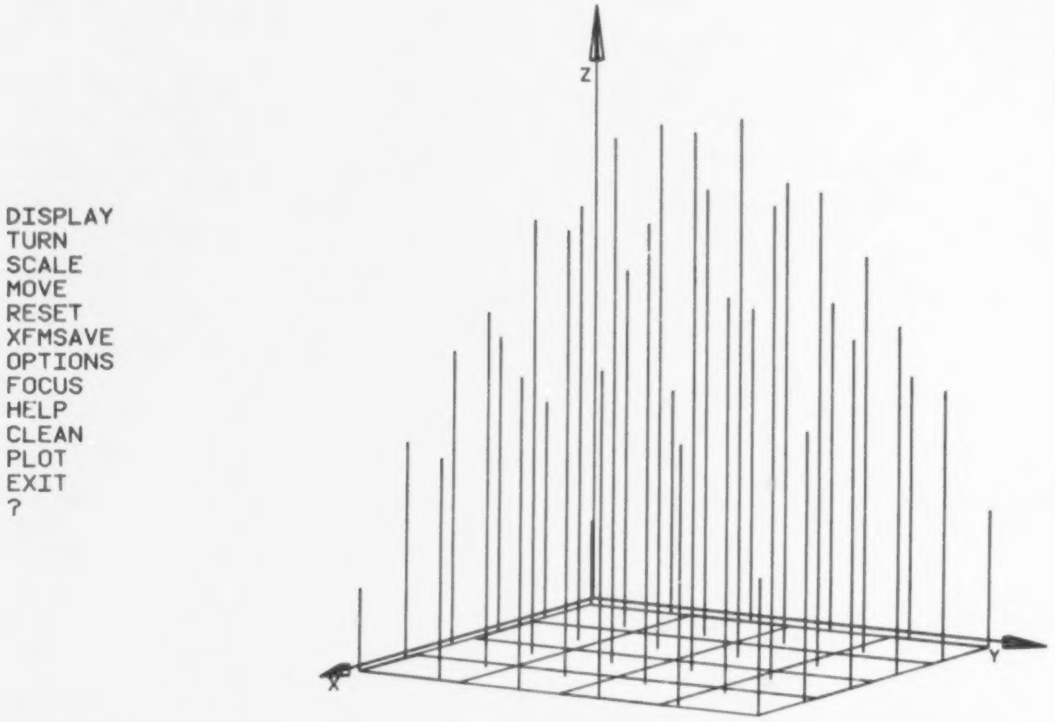


Figure 2. NEEDLES and XYPLANE options on.

BOX (B)

Restrictions: None

Default: OFF

ON: The procedure displays the data within a box extending from the minimum to the maximum data value along each of X, Y, and Z axes.

OFF: The box is not shown.

ORIGIN (O)

Restrictions: None

Default: ON

ON: DRAW3D displays the XYZ axes with their origin at the point [0.,0.,0.] in XYZ space.

OFF: The axes originate at the point that is the minimum of the range of your data.

STEREO (S)

Restrictions: None

Default: OFF

ON: DRAW3D shows two images of the data, one for each eye. The images are rotated slightly with respect to each other so that you can fuse them stereoscopically to produce an image with depth. The position of the two images on the screen depends on the state of the VERTICAL option (described below). A line is drawn across the screen between the two images to help you find the correct orientation to fuse them. If the AXES option is ON, the axes are shown but are not labeled.

OFF: Only one image is shown.

VERTICAL (V)

Restrictions: The STEREO option should be ON before you change the VERTICAL option. The VERTICAL option has no effect unless the STEREO option is on.

Default: ON

ON: If the STEREO option is OFF, this option has no effect. When STEREO is ON, the procedure displays two images one above the other, with the upper one upside down. This stereo pair is suitable for fusing with the Amsom stereoscopic apparatus ("Amsom box"). Note: For information about obtaining an Amsom box, contact Eric Meyer by system mail or on the PROPHET line.

OFF: If the STEREO option is OFF, this option has no effect. When STEREO is ON, DRAW3D displays the two images side by side, both right side up. This pair is suitable for fusing with conventional stereoscopic lenses, or with the naked eye.

LABELS (L)	Restrictions: The AXES option must be ON to change the LABELS option. Default: OFF ON: When the AXES option is OFF, the LABELS option has no effect. When AXES is ON, each axis is labeled with the numeric value along that axis at the origin, the maximum data value, and 1/4, 1/2, and 3/4 of the way to the maximum value. OFF: When the AXES option is OFF, the LABELS option has no effect. When AXES is ON, each axis is labeled only with its name (X, Y, or Z) and not with numeric values.
CMDS (C)	Restrictions: None Default: ON ON: After drawing the display as it currently appears, DRAW3D prompts you with a list of commands. If you choose the OPTIONS command, it further prompts you with a list of options. OFF: After drawing the display, DRAW3D prompts you only with a question mark. If you choose the OPTIONS command, it then prompts you with a double question mark (??) to indicate that it expects an option rather than a command.
CRTL-E	This is not really an option, but is included in case you specify the OPTIONS command by mistake. If you give the OPTIONS command and decide you do not want to change any of the options, CRTL-E causes DRAW3D to ask for another command without changing any of the options.
Advanced DRAW3D	This section is concerned with the use of the XFMSAVE (X) command and the two entries REDRAW3D and D3DQT. You can use these to save the appearance of a display so that it can be retrieved quickly later, and to see more than one object on the screen at the same time. What DRAW3D shows on the screen is defined entirely by four PROPHET variables: the display matrix of data points, two PROPHET transformers, and a boolean array of options.

The Display Matrix

The display matrix has three columns and many rows, each row representing one (X,Y,Z) point. DRAW3D generates the display you see when the GRID option is ON by simply connecting each of these points to the next point with a line.

If the same point is to be visited more than once, it must occur more than once in the matrix. In general, if the number of X values is m and the number of Y values is n, there will be 2mn rows in the display matrix (or, if m and n are not even numbers, 2(m+1)(n+1) rows).

The Transformers

The two transformers define the state of the surface coordinate system and the screen coordinate system. A transformer is a FLOAT ARRAY ARRAY of 4 rows and 3 columns, whose meaning is discussed in depth in Chapter XII of the <u>User's Guide to PL/PROPHET</u>.

Basically, the "raw" XYZ points in the display matrix are transformed by the surface transformer and then by the screen transformer to arrive at their coordinates in UVW space. (Actually, there is also a final, invariant transformation of scaling by 100 to get into real terminal coordinates.) Then the transformed version of the data is drawn on the screen, according to the status of the various options.

The Boolean Array of Options

The boolean array of options tells DRAW3D whether each of the options (GRID, NEEDLES, etc.) is ON or OFF (TRUE or FALSE). The ordering of the options in this array is the same as the ordering of the options when they appear on the screen. Thus, the default options array is:

```
[TRUE, (GRID)
FALSE, (NEEDLES)
FALSE, (PERSPECTIVE)
TRUE, (AXES)
FALSE, (XYPLANE)
FALSE, (BOX)
TRUE, (ORIGIN)
FALSE, (STEREO)
TRUE, (VERTICAL)
FALSE, (LABELS)
TRUE] (CMDS)
```

XFMSAVE Command

The purpose of the XFMSAVE (X) command is to get a top-level copy of the two transformers and the options array. This is desirable when you have manipulated the display to the appearance you find most useful. When you invoke the XFMSAVE command, you are asked to supply top-level names for the transformers and the option array. Then you can use the entries REDRAW3D and D3DQT to bypass all your previous work and see the display immediately as it was when you used XFMSAVE.

REDRAW3D Entry

REDRAW3D takes four arguments: the display matrix, the surface transformer, the screen transformer, and an array defining the current state of the options. REDRAW3D causes you to rejoin the regular DRAW3D interaction. It thus enables you to change the display further, use XFMSAVE, and do anything you could do in regular DRAW3D.

D3DQT Entry

D3DQT takes the same four arguments as REDRAW3D, plus a fifth: a DRAW3D command or command abbreviation. D3DQT does nothing whatever except execute this command on the surface defined by its first four arguments. It does not display the surface unless you explicitly specify DISPLAY as the command argument, and even then it does not erase the screen first. It does not interactively ask for any commands, but it will ask you for a particular display option if you specify OPTIONS as the fifth argument.

D3DQT is useful if you wish to display more than one set of data on the screen at the same time. You can, for example, display a set of data points together with a surface statistically fitted through them. In general, it is probably true that when you display more than one surface at once, you will want to use the same two transformers for both surfaces, or at least transformers that are logically related. The options array may be varied to aid understanding.

By referring to the PROPHET User's Manual, you can learn to manipulate the display transformers directly without going through DRAW3D a second time. However, it is probably clearer and simpler to use DRAW3D or REDRAW3D.

Warning!	When using REDRAW3D and D3DQT, be sure to supply the two transformers in the correct order: first the surface transformer, and then the screen transformer. Reversing them produces a display that is wildly different from the one you saw when you used XFMSAVE.

MULT3D

by Eric Meyer
Bolt Beranek and Newman Inc.

Summary

MULT3D is a generalization of the public procedure DRAW3D; it makes it easy to display more than one surface at the same time.

Calling Sequence

```
RAC PUBLIC MULT3D <GO>
CALL MULT3D <GO>
```

Citation

Meyer EJ. MULT3D, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Contact Eric Meyer on the PROPHET line.

Things to Remember

If you are not already familiar with the public procedures GRAPH3D and DRAW3D, you should read their descriptions in this notebook before before trying to understand this discussion of MULT3D.

MULT3D works only on display terminals.

MULT3D is an interactive procedure that enables you to display and manipulate several three-dimensional surfaces at the same time. It has commands almost identical to the commands in DRAW3D, except that in the context of MULT3D the commands can refer either to an individual surface or to the whole system of surfaces you are working with. A surface is defined as one FLOAT ARRAY ARRAY of [X,Y,Z] points, having three columns and any number of rows, representing one indivisible group of data points. The "system" is simply the set of all surfaces currently defined.

Defining the Data Points

When you call MULT3D, you must define the data points for the first surface you want to display. If you used GRAPH3D to make a display matrix, MULT3D lets you use this matrix. Otherwise, it invokes GRAPH3D to specify the points however you like.

Fitting the Points into the System

Once you have the data points for the new surface, MULT3D provides several ways to fit them into the system. A surface's orientation and appearance in the system are governed by two transformers (surface and screen) and a boolean array of options (representing such things as whether or not the axes are to be shown).

You might have created transformers and an option array by using the XFMSAVE command in a previous run of DRAW3D or MULT3D; you might also have made them directly, by hand. If you do have transformers and an option array and you want to apply them to the surface you are now creating, MULT3D lets you do so.

If you do not have a pair of transformers and an option array that you would like to apply to your new surface, MULT3D invokes DRAW3D. You can now use DRAW3D's commands to move the surface and change its options until it looks like you want it to look in your system of surfaces. Then type the EXIT command; you will return to MULT3D with the surface fully defined.

ADD Command

You can use the ADD command to add another surface to your system.

When you give this command, you enter into an interactive quiz to help you define the new surface. This quiz is just like the one you deal with when you initially call MULT3D to define your first surface.

Only one question is different: when you use the ADD command, MULT3D asks whether you want the new surface to be set up like a previous one. If you do, and if there are already two or more in the system, you must specify which previous surface the new one should be modeled after. MULT3D then applies that old surface's tranformers and option array to the new surface and displays the entire system.

DELETE Command

If you have a system of surfaces and you need to eliminate one without disturbing the rest or starting over, the DELETE command enables you to delete one surface.

When you issue the DELETE command, MULT3D asks you for the number of the surface you want to delete. Then, before actually deleting anything, it redisplays the surface whose number you gave and asks whether that was the one you meant. Thus, if you make a mistake, you can easily recover without losing the wrong surface.

Note that after it deletes a surface, MULT3D renumbers the higher-numbered surfaces to fill in the gap. For example, if you have three surfaces and you delete the second one, the one that was originally number 3 becomes the new number 2.

Other MULT3D Commands

The rest of the MULT3D commands correspond to the commands available in DRAW3D, with a few notable differences:

- When you give certain commands in MULT3D, the procedure responds by asking:

  ```
  SURFACE?
  ```

  You should type the number of the surface to which you want your command to apply. (If you forget how the surfaces are numbered, give the DISPLAY command; MULT3D displays the surfaces in numerical order.) Respond with CTRL-E if you want the command to be executed for every surface in the system.

- You can spatially tranform the entire system of surfaces by first giving a TURN, SCALE, or MOVE command and then typing CTRL-E in response to the SURFACE? prompt. However, you can do so only in the screen coordinate system (UVW space), because the various surface coordinate systems (the XYZ spaces) may not be consistent.

- In DRAW3D you can abbreviate any command by typing only its first letter. In MULT3D, both

DISPLAY and DELETE are valid commands, so to avoid ambiguity you must type out at least two letters. You can still abbreviate the remaining commands with single letters.

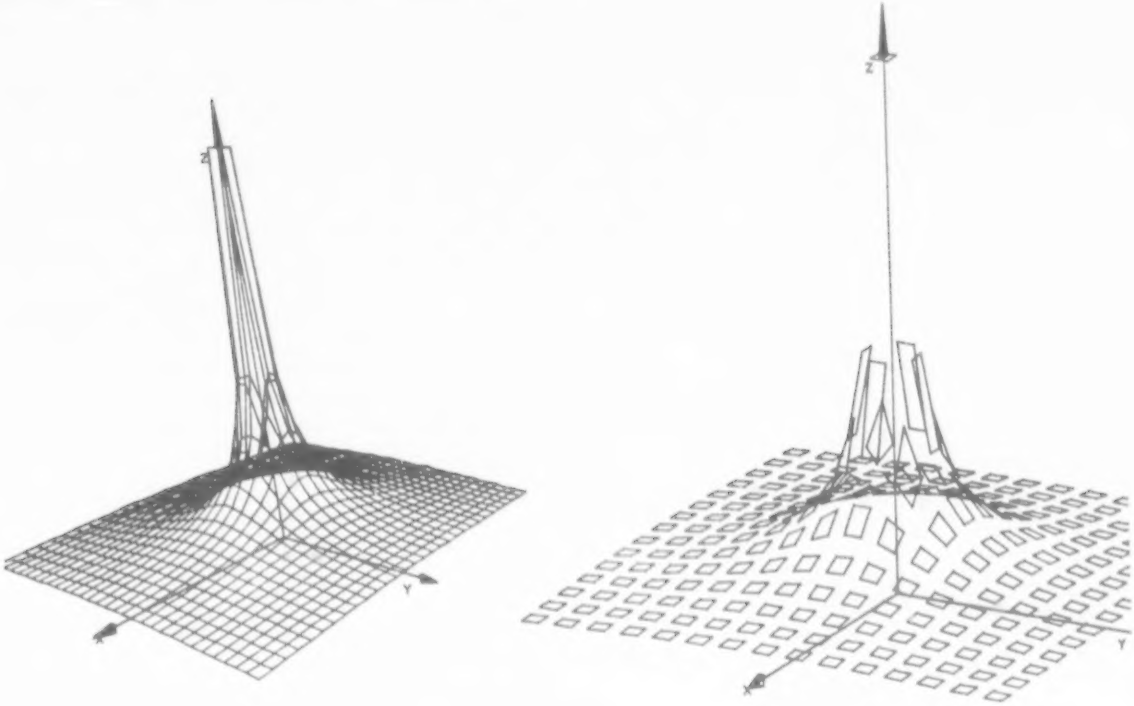


Figure 1. An example of a multiple-display created using MULT3D.

BARGRAPH

by
James Jay Wood
and
Harold Perry
Bolt Beranek and Newman Inc.

Summary

Using percentile data stored row-wise in a table, BARGRAPH constructs a graph consisting of groups of bars for each row. If your data are not percentages, BARGRAPH converts them into percentages to produce a BARGRAPH. (The raw data remain unchanged in your table.)

Calling Sequence

```
RAC PUBLIC BARGRAPH <GO>
CALL BARGRAPH(data-table)
               <table>
```

Entries

```
CALL BARGRAPHQT(data-table, percentile-data,  colspecs,
                 <table>         <bool>      <fixed array>

              barspecs)
            <text array>


CALL BARGRAPHCONST(data-table, percentile-data,  colspecs,
                    <table>         <bool>      <fixed array>

                 barspecs)
               <text array>
```

Citation

No citation necessary.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things to Remember

The BARGRAPH public procedure is different from the MAKE BARGRAPH command described in the PROPHET User's Manual.

The input table must not contain empty cells. If your table contains percentile data, the values must be between 0 and 100, inclusive.

The graph that BARGRAPH constructs is not an ordinary PROPHET graph; it cannot be moved, scaled, etc. However, once the bar graph is displayed, you may CLEAN the screen, plot the graph, and/or make a hard copy of the graph.

BARGRAPH works only on display terminals.

Interaction with BARGRAPH

BARGRAPH enables you to construct well-formatted bar graphs of percentile data. To call BARGRAPH, use the command

```
CALL BARGRAPH(data-table)
```

where data-table is the name of your table of data.

The procedure first asks whether your table contains percentile data. If it does, BARGRAPH uses these values as the heights of the bars. If not, the procedure converts the raw data to percentages for you and uses these for the bar heights. The raw data are not changed to percentages in your PROPHET table.

BARGRAPH next asks you to indicate the columns you want to include in each cluster of the bar graph and how each column bar should be displayed. (See the description of BARGRAPHQT or the example below for a list of the possible ways to display column bars.) When BARGRAPH computes percentile data for you, it uses the number of columns you specify, not the total number of columns in your data table.

BARGRAPH then constructs and displays a bar graph subject to your specifications. The graph's label is the title of your data table. However, because the procedure graphs your data as percentages, the title of your data table may not be an appropriate label for the graph; consult the example below for illustration. (You can set the graph's label by setting the TITLE of your data table before calling BARGRAPH.) The bar graph also includes a legend that correlates bar labels with bar display options.

The X axis is labeled with the heading of column 0 of your data table. (You can set this label by setting the HEADING of column 0 of your data table before calling BARGRAPH.)

Quiet Entry, BARGRAPHQT

BARGRAPH has a quiet entry, BARGRAPHQT. It constructs a bar graph automatically, subject to the arguments shown below in Table 1.

Table 1. BARGRAPHQT Arguments.

Argument	Type	Description
data-table	<table>	Table with data for bar graph
percentile-data	<bool>	If TRUE, data are treated as percentages. (Be sure that the values range from 0. to 100., inclusive.) If FALSE, the procedure converts to percentile data before producing bar graph.
colspecs	<fixed array>	Indices of the columns of data-table to be included
barspecs	<text array>	Type of bar to be used. Barspecs[n], indicates how the data in colspecs[n] will be displayed.

best copy available

Each element of barspecs must be one of the following:

'P' for Plain
'S' for Solid
'R' for Right-Hatch
'L' for Left-Hatch
'X' for Cross-Hatch
'H' for Horizontal Lines
'V' for Vertical Lines
'C' for Checkerboard
'B' for Herringbone

For example, the following entry would produce a bar graph containing a group of bars for each row of BTAB:

```
BARGRAPHQT(BTAB,TRUE,[1,3,5],['P','L','R'])
```

Each group would contain three bars. The height of the first bar in each group would be equal to the value from column 1; that of bar 2, to the value from column 3; that of bar 3, to the value from column 5. The first bar in each group would be plain; the second, left-hatched; the third, right-hatched.

BARGRAPHCONST Entry

BARGRAPHCONST is similar to BARGRAPHQT except that it asks for other columns to be included in the graph in addition to those specified by the arguments. (Note that the arguments are identical to those used by BARGRAPHQT.) It displays the data in the columns specified in colspecs, as well as the data in any other columns you specify interactively. This entry is particularly useful when you want to make a set of bar graphs containing bars for a common subset of column specifications.

RSRCH: PERFORMERS OF BASIC RESEARCH (IN $MILLIONS) 6R X 4C

0 YEAR	1 FEDERAL GOVERNMENT	2 UNIVERSITIES	3 INDUSTRY	4 (D) TOTAL
1. 1958	24.	49.	40.	113.
2. 1960	22.	56.	38.	116.
3. 1962	23.	60.	37.	120.
4. 1964	24.	67.	35.	126.
5. 1966	25.	72.	32.	129.
6. 1968	25.	78.	31.	134.

Figure 1. A typical data table for BARGRAPH.

```
>DIS RSRCH<

>RAC PUBLIC BARGRAPH<
>CALL BARGRAPH(RSRCH)<

DOES RSRCH CONTAIN PERCENTILE DATA?NO<

PLEASE SPECIFY COLUMNS TO BE INCLUDED IN THE BAR GRAPH
PLUS EACH COLUMN DISPLAY OPTION;

CHOICES FOR COLUMN DISPLAY OPTIONS INCLUDE:

        PLAIN  (P),
        SOLID  (S),
        RIGHT-HATCH (R),
        LEFT-HATCH  (L),
        CROSS-HATCH (X),
        HORIZONTAL LINES (H),
        VERTICAL LINES (V),
        CHECKERBOARD (C),
        HERRINGBONE (B).

WHEN FINISHED, RESPOND WITH "0" TO THE "COLUMN #" PROMPT.

COLUMN # 1 = ?1< DISPLAY OPTION?X<
COLUMN # 2 = ?3< DISPLAY OPTION?P<
COLUMN # 3 = ?2< DISPLAY OPTION?H<
COLUMN # 4 = ?0<
```

Figure 2. Interacting with BARGRAPH.

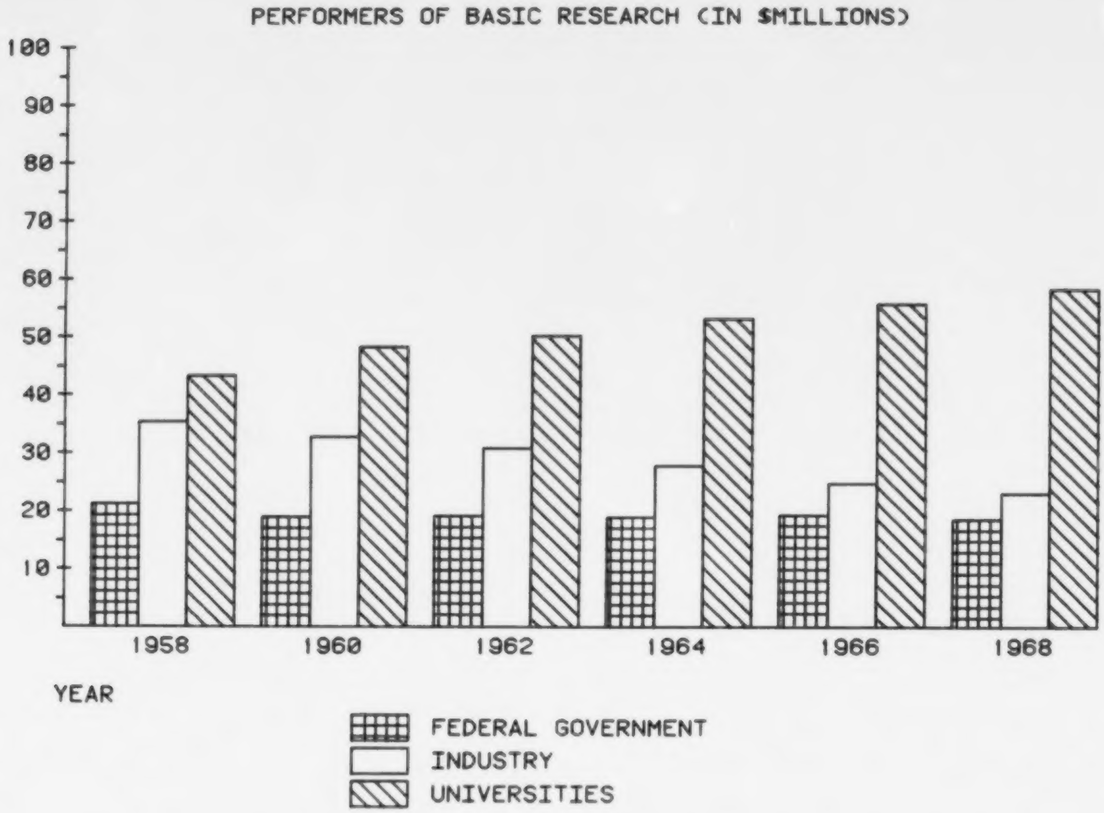


Figure 3. BARGRAPH's output.

XFROMY

by Harold Perry
Bolt Beranek and Newman Inc.

Summary

Given a Y value from a curve F of a graph, XFROMY computes an X value such that F(X) = Y. F must be either a fitted curve or a curve added as a function to the graph. If appropriate, you may specify the pharmaceutical parameters ED_n, LD_n, or I_n (where $0<n<100$) as the value for Y. You can also obtain a temporary table with all computed values.

Calling Sequence

```
RAC PUBLIC XFROMY <GO>
CALL XFROMY <GO>
```

Entries

```
CALL XFROMYPOLY <GO>

CALL XFROMYQT (data-graph, curve-index, Y-tab,
                <graph>    <fixed>    <table>

               low-X,    high-X,   sig-digits)
              <float>    <float>     <fixed>
```

Citation

Perry HM. XFROMY, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge MA, January 1982.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things to Remember

Use the XFROMY entry only if your curve is monotonic, i.e., the slope of the curve never changes in sign. If the curve is not monotonic, use XFROMYPOLY or XFROMYQT.

XFROMY's Computations

Given a value Y from a curve F, XFROMY computes that X value such that F(X) = Y. Your curve F must be a fitted curve added to your graph or a curve added to your graph as a function.

XFROMY works only if F is monotonic: the slope must be either always positive or always negative throughout the entire graph (i.e., F(X) is either always rising or always falling for increasing X values). If your curve F is not monotonic, you must use the XFROMYPOLY entry or the quiet XFROMYQT entry.

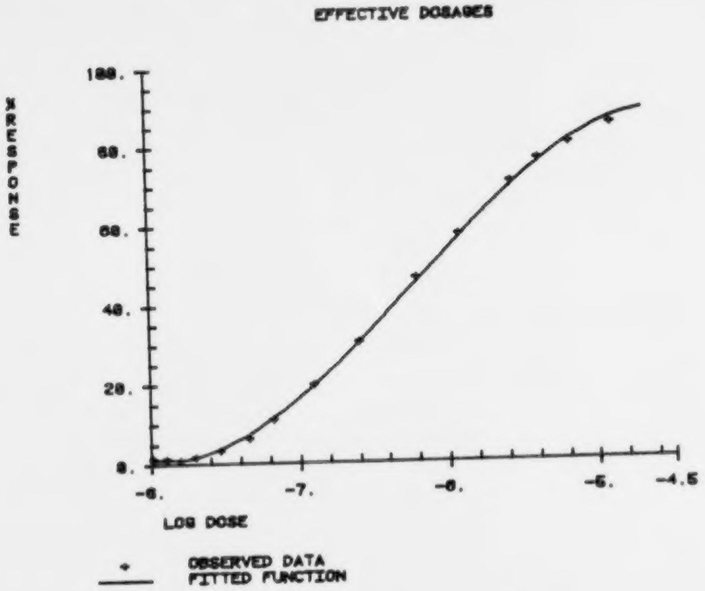


Figure 1. A monotonic dose-response curve from which X values are to be calculated.

In many applications, you may not know the numeric value of a Y for which you want to compute an X. For example, you may have a drug dose-response curve and want to know the dose at which 50% of the drug was absorbed, i.e., the ED50 of the drug. XFROMY solves these problems by allowing you to specify ED_n, LD_n, or I_n ($0<n<100$) for Y values. (ED represents effective dose, LD represents lethal dose, and I represents immission.)

If you specify ED_n, LD_n, or I_n as Y, where $0<n<100$, then XFROMY computes X such that

$$F(X) = Y_{\text{low}} + \frac{n}{100} \cdot (Y_{\text{high}} - Y_{\text{low}})$$

where Y_{low} is the smallest value of F over the range of X values in the curve, and Y_{high} is the largest. (Often, these values are just 0.0 and 1.0, respectively.)

Interactive Entry for Monotonic Curves (XFROMY)

If your curve F is monotonic, then you can interactively compute X values by using the XFROMY entry. Once called, XFROMY asks you to supply the following:

- the name of the graph containing your data
- the index of the curve F
- the number of significant digits you want the computed X values to have

XFROMY then computes and reports an X value for each Y value that you supply (see Figure 2). If the supplied Y value is not on F, the procedure ignores your request.

```
>RAC PUBLIC XFROMY<
>CALL XFROMY< NAME OF GRAPH CONTAINING DATA?EDGRAPH<

INDEX OF CURVE FROM EDGRAPH FOR
WHICH YOU WANT TO COMPUTE X VALUES?2<
ENTER NUMBER OF SIGNIFICANT FIGURES DESIRED?5<

ENTER Y VALUE (WHEN DONE, HIT <GO>):50<
F(-6.1067) = 50.
ENTER Y VALUE (WHEN DONE, HIT <GO>):ED50<
F(-6.3076) = ED50
ENTER Y VALUE (WHEN DONE, HIT <GO>):ED25<
F(-6.8517) = ED25
ENTER Y VALUE (WHEN DONE, HIT <GO>):ED75<
F(-5.7802) = ED75
ENTER Y VALUE (WHEN DONE, HIT <GO>):10<
F(-7.2387) = 10.
ENTER Y VALUE (WHEN DONE, HIT <GO>):<
```

Figure 2. Using XFROMY to compute X values for the graph displayed in Figure 1.

When you have finished inputting Y values, you can make a temporary table containing all the X and Y data; XFROMY sorts the X values in this table. Figure 3 illustrates this option.

```
WOULD YOU LIKE THE X AND Y VALUES SAVED IN
A TEMPORARY TABLE?YES<
WHAT NAME DO YOU WANT FOR THE TABLE ?EDDATA<

>DIS EDDATA<
```

INTERPOLATED X VALUES FROM
CURVE 2 OF EDGRAPH 5R X 2C

0	1 X	2 Fitted Function
1.	-7.2387	10.
2. ED25	-6.8517	21.62201
3. ED50	-6.3076	42.05632
4.	-6.1067	50.
5. ED75	-5.7602	62.49062

Figure 3. Saving XFROMY's results in a temporary table.

Interactive Entry for Nonmonotonic Curves (XFROMYPOLY)

If your curve F is not monotonic, then to compute X values interactively, you must use the XFROMYPOLY entry. For the inverse prediction algorithm to work, you must also supply low-X and high-X values that define a monotonic portion of F. Make sure that the Y values you input correspond to the curve portion of F you specified. You can change the low-X and high-X values at any time to perform inverse prediction on a different monotonic portion of F. You can use ED_n, LD_n, or I_n to specify Y values in both XFROMY and XFROMYPOLY.

Quiet Entry XFROMYQT

The quiet entry to the procedure, XFROMYQT, requires the arguments shown in Table 1.

Table 1. XFROMYQT Arguments.

Name	Type	Description
data-graph	<graph>	Your graph with a function curve F
curve-index	<fixed>	Index of the curve F
Y-tab	<table>	Table containing Y values of F for which X values should be computed (see below)
low-X, high-X	<float> <float>	If F is monotonic, these should both be EMPTY; otherwise, they should be X values that define a monotonic portion of F
sig-digits	<fixed>	Desired number of significant digits in the computed X values (must be between 1 and 7, inclusive)

If any of your Y values are of the form ED_n, LD_n, or I_n ($0<n<100$), then the entries of Y-tab must be in column 0 and not in any other column. (Thus, when you make Y-tab, the row names serve as the Y values for which you want to compute X values.) You can mix numeric Y values with those of the above form.

The procedure returns two new columns of numeric data to Y-tab: column 1 will contain computed X values; column 2 will contain the given Y values. Therefore, any Y values in column 0 of form ED_n, LD_n, or I_n will have their equivalent numeric values in column 2. Figure 4 shows an example of how Y-tab might look before and after the call to XFROMYQT.

```
>DIS YTAB<



YTAB:  THIRD ARGUMENT
FOR XFROMYQT  5R X 0C
+----------+
|0         |
|ROWNAME   |
+----------+
|1. 50     |
|2. ED50   |
|3. ED25   |
|4. ED75   |
|5. 10     |
+----------+



>RAC PUBLIC XFROMY<
>CALL XFROMYQT(EDGRAPH,2,YTAB,,,5)<
>DIS YTAB<



YTAB:  THIRD ARGUMENT
FOR XFROMYQT  5R X 2C
+----------++---------+----------+
|0         ||1        |2         |
|ROWNAME   ||X        |Y         |
+----------++---------+----------+
|1. 50     ||-6.1067  |50.       |
|2. ED50   ||-6.3076  |42.05632  |
|3. ED25   ||-6.8517  |21.62201  |
|4. ED75   ||-5.7802  |62.49062  |
|5. 10     ||-7.2387  |10.       |
+----------++---------+----------+
```

Figure 4. Using the quiet entry, XFROMYQT.

If none of your Y values is of the form ED_n, LD_n, or I_n, then Y-tab should not have row names. Instead, it should contain all Y values in column 1, which should be of type NUMBER. The procedure returns a new column 1 containing the computed X values. Column 2 then contains the given Y values.

If any Y values are EMPTY or out of the range of the specified curve, XFROMYQT ignores them. XFROMYQT does not sort Y-tab.

3. Numerical Procedures

3. Numerical Procedures

3. Numerical Procedures

ROOT	finds real and complex roots of a polynomial .	3-1
EIGEN	computes eigenvalues and eigenvectors of a real symmetric matrix	3-5
INTERPOLATE	performs cubic spline interpolation	3-9
SMOOTH	smooths experimental data.	3-17
PLANIM	calculates the area within a closed curve . .	3-21
INTEGRAL	finds the area under or bounded by a curve . .	3-23
TRAP	finds the area under a curve using the trapezoidal rule	3-27
STEPAREA	finds the incremental area under an exponential decay curve using the cubic spline method . .	3-31

ROOT

by Jean Risley
Bolt Beranek and Newman Inc.

Summary

ROOT finds the real and complex roots of a polynomial.

Calling Sequence

```
RAC PUBLIC ROOT <GO>
CALL ROOT(coeff-array,  real-array,   imag-array)
        <float array> <float array> <float array>
```

Citation

Risley J. ROOT, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Contact Margaret Reid-Miller on the PROPHET line.

Things to Remember

You must declare all arrays before calling ROOT.

Coeff-array must contain no empty or missing values.

ROOT is a procedure that finds the real and complex roots of a polynomial when provided with an array of coefficients. The coefficients of the polynomial must be real.

ROOT's Arguments and Output

To use ROOT, type the command:

```
CALL ROOT(coeff-array, real-array, imag-array)
```

where each of the three arguments is a one-dimensional FLOAT ARRAY. Coeff-array must contain the input values of the coefficients of the polynomial. The first element in coeff-array should be the constant (or zero-order coefficient), and the coefficients should be in order, ending with the coefficient of the highest power. Real-array receives the values of the real part of the roots, and imag-array receives the values of the imaginary part of the roots. Each root has both a real and an imaginary part; the procedure returns zeroes, rather than empties, when one of the parts does not apply.

If the procedure is unable to compute all of the roots correctly, it types the number of the root that failed to converge. It provides the root values for the roots that did converge in the output arrays.

Reference

Muller DE. A method for solving algebraic equations using the automatic computer. Mathematical Tables and Other Aids to Computation 10:208, 1956. (contains a description of the technique used to compute the roots)

Example

The roots of the polynomial

$$1 - 5X + 7X - 5X + 6X$$

are at X = 1/2, X = 1/3, X = i, and X = -i. Figure 1 shows how you would use ROOT to calculate them.

```
>DCL (COF,RE,IM) FLOAT ARRAY<
>COF(1)=1.<
>COF(2)=-5.<
>COF(3)=7.<
>COF(4)=-5.<
>COF(5)=6.<
>RAC PUBLIC ROOT<
>CALL ROOT(COF,RE,IM)<
>TYPE RE< .5
.3333333
0.
0.

>TYPE IM< 0.
0.
1.
-1.
```

Figure 1. Using ROOT.

EIGEN

by
Jack Fitzpatrick
and
David Fram
Bolt Beranek and Newman Inc.

Summary

EIGEN computes eigenvectors and eigenvalues of a real, symmetric matrix.

Calling Sequence

```
RAC PUBLIC EIGEN <GO>
CALL EIGEN(  matrix,    eigen-vals,    eigen-vecs)
          <float array <float array> <float array
             array>                     array>
```

Citation

No citation necessary.

Questions or Comments

Contact Margaret Reid-Miller on the PROPHET line.

Things to Remember

You must declare all arrays before calling EIGEN.

The maximal size for matrix is 50 by 50.

There must be no EMPTY values in matrix.

The system procedure %EIGENQR (which requires the same arguments as EIGEN) computes eigenvectors and eigenvalues significantly faster than EIGEN for matrices no larger than 7 by 7.

The public procedure EIGEN runs a FORTRAN program taken from First Data Corporation's Scientific Subroutine Library. The FORTRAN program uses the diagonalization method of Jacobi as adopted by von Neumann.

Calling EIGEN

To use EIGEN, give the command:

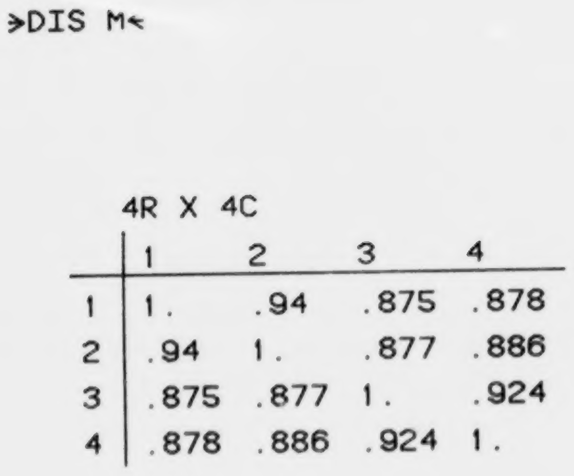
```
CALL EIGEN(matrix, eigen-vals, eigen-vecs)
```

The first argument is the name of the real, symmetric matrix under consideration. The second and third arguments are, respectively, a FLOAT ARRAY and a FLOAT ARRAY ARRAY that you must declare before calling the procedure. When EIGEN is finished, eigen-vals contains the eigenvalues in order of decreasing size, and eigen-vecs contains the eigenvectors, stored by row, in the same order.

Consult figures 1 through 4 for an example of using EIGEN.

Reference

Ralston A and Wilf HS, eds. Mathematical Methods for Digital Computers Chapter 7. Wiley, New York, 1962.



```
>DIS M<


      4R X 4C
    | 1     2     3     4
  --+----------------------
  1 | 1.    .94   .875  .878
  2 | .94   1.    .877  .886
  3 | .875  .877  1.    .924
  4 | .878  .886  .924  1.
```

Figure 1. A typical matrix for EIGEN.

```
>DCL VALUES FLOAT ARRAY<
>DCL VECTORS FLOAT ARRAY ARRAY<
>RAC PUBLIC EIGEN<
>CALL EIGEN(M,VALUES,VECTORS)<
```

Figure 2. Using EIGEN.

```
>DIS VALUES<


     4R X 1C
        | 1
     ---+----------
      1 | 3.690027
      2 | .1742662
      3 | .07641099
      4 | .05929613
```

Figure 3. Computed eigenvalues.

```
>DIS VECTORS<


     4R X 4C
    | 1          2          3          4
  --+-------------------------------------------
  1 | .5004482   .5018238   .4980277   .4996928
  2 | -.5193034  -.4770674  .5302524   .4707048
  3 | -.150694   .1136321   -.6769091  .7114585
  4 | -.6761412  .7125077   .112227    -.150236
```

Figure 4. Computed eigenvectors.

INTERPOLATE

by

Harold Perry
Bolt Beranek and Newman Inc.
and
Karl Kuhlmann
Stanford Research Institute

Summary

INTERPOLATE performs a cubic spline interpolation on a set of data.

Calling Sequence

```
RAC PUBLIC INTERPOLATE <GO>
CALL INTERPOLATE <GO>
```

Entry

```
CALL MULTINTERPOLATE <GO>
CALL INTERPOLATEQT(function-data, ind-var-col, func-val-col,
                     <table>       <fixed>       <fixed>
                                or
                <float array array>

                 interp-data, interp-ind-var-col, extrap)
                   <table>          <fixed>        <bool>
                               or
                <float array array>

CALL MULTINTERPOLATEQT (table-names, ind-var-col, func-val-col,
                        <text array>    <fixed>      <fixed>

                        interp-tab, interp-ind-var-col, extrap)
                         <table>          <fixed>        <bool>
```

Citation

Perry HM and Kuhlmann K. INTERPOLATE, in *PROPHET Public Procedures Notebook* (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Question or Comments

Contact Harold Perry on the PROPHET line.

Introduction

INTERPOLATE calculates function values for functions that are not expressed in closed form. Given a set of X values and a corresponding set of Y values, INTERPOLATE forces a cubic spline function through these data pairs and uses this function to calculate function values for new X values.

The cubic spline function is a piece-wise function of cubic polynomials forced through each set of two contiguous data points, with the restriction that the polynomials to the left and right of a given data point must have the same value and the same first and second derivatives at that data point. Thus, the polynomials join smoothly at each data point with the value of that data point and have the same slope and curvature.

The spline function from minus infinity to the first data point and from the last data point to infinity is assumed to be linear, with the same slope as the first or last data interval. Hence, extrapolated values for X values outside the range of the original set are not as accurate as interpolated values for X values inside the range.

Interaction with INTERPOLATE

To use INTERPOLATE, type the command:

```
CALL INTERPOLATE <GO>
```

The procedure asks the following question:

```
NAME OF SOURCE TABLE OR FLOAT ARRAY ARRAY
WHICH CONTAINS X AND Y DATA?
```

Your response should be the name of the table or FLOAT ARRAY ARRAY that contains the set of X values and their corresponding Y values. (These entries need not be sorted.) The procedure then asks:

```
COLUMN OF YOUR DATA WITH X VALUES?

COLUMN OF YOUR DATA WITH Y VALUES?
```

Your responses to these questions should be the fixed column indices of your data which contain X and Y values, respectively.

```
NAME OF DESTINATION TABLE OR FLOAT ARRAY ARRAY
WHICH CONTAINS X VALUES TO BE INTERPOLATED?

COLUMN OF YOUR INTERPOLATION DATA WITH X
VALUES TO BE INTERPOLATED?
```

Your response should be the name of the table or FLOAT ARRAY ARRAY and a column of X values for which you want to interpolate (i.e., compute their corresponding function values), using the function data specified by your previous responses. Finally, the procedure asks:

```
DO YOU WANT TO PERFORM EXTRAPOLATION FOR X
VALUES IN THE DESTINATION DATA THAT MAY BE
OUT OF RANGE OF THE X VALUES FROM THE
SOURCE DATA?
```

If you answer "YES", such X values will be automatically extrapolated. Otherwise, they will be ignored.

After you supply proper responses, INTERPOLATE performs the specified interpolation and adds a new column of interpolated values to your interpolation data. It then displays your new interpolation data. Consult Figures 1 through 4 for details.

MULTINTERPOLATE Entry

The MULTINTERPOLATE entry is designed to perform a series of interpolations for a given set of X values. A summary table with the pooled interpolation results will be provided. To use this entry, type the command:

```
CALL MULTINTERPOLATE <GO>
```

This entry insists that your X, Y data be stored in tables. Moreover, the X and Y data must be saved in the same columns for each table.

After you specify this information, MULTINTERPOLATE proceeds in a manner identical to INTERPOLATE. Next, indicate the table with interpolation data and whether extrapolation should be performed. MULTINTERPOLATE will then append the interpolation results for each set of X, Y data to the destination table. Refer to Figures 5 through 8 for further information.

Quiet Entry INTERPOLATEQT

INTERPOLATE also has a quiet entry, INTERPOLATEQT. To use this procedure, issue the command:

```
CALL INTERPOLATEQT(function-data, ind-var-col,
                   func-val-col, interp-data,
                   interp-ind-var-col, extrap)
```

The arguments to INTERPOLATEQT correspond exactly to the responses you give to the six questions INTERPOLATE asks.

Once invoked, INTERPOLATEQT adds a new column of interpolated values to your interpolation data. It does not ask you any questions, and it does not display the results.

Quiet Entry MULTINTERPOLATEQT

Finally, there is a quiet version, MULTINTERPOLATEQT, of the MULTINTERPOLATE entry. This entry performs the same computations, but is not interactive. To use this entry, type the command:

```
CALL MULTINTERPOLATEQT (table-names, ind-var-col,
                       func-val-col,interp-tab,
                       interp-ind-var-col,
                       extrap)
```

The arguments to MULTINTERPOLATEQT correspond exactly to the responses you provide to MULTINTERPOLATE.

After completion of MULTINTERPOLATEQT, the pooled interpolation results will be appended to the interp-tab table.

ORIGDATA 10R X 2C

	1 X	2 Y
1	.1	2.734490
2	.2	2.837959
3	.3	3.010408
4	.4	3.251837
5	.5	3.562245
6	.6	3.941633
7	.7	4.390000
8	.8	4.907347
9	.9	5.493673
10	1.0	6.148980

Figure 1. Typical function data.

NEWDATA 9R X 1C

	1 NEW X
1	.325
2	.645
3	.865
4	.234
5	.111
6	.976
7	1.
8	1.1
9	.1

Figure 2. Interpolation data.

```
>RAC PUBLIC INTERPOLATE<
>CALL INTERPOLATE<

NAME OF SOURCE TABLE OR FLOAT ARRAY ARRAY WHICH CONTAINS
X AND Y DATA?ORIGDATA<

COLUMN OF ORIGDATA CONTAINING
X VALUES?1<

COLUMN OF ORIGDATA CONTAINING
Y VALUES?2<

NAME OF DESTINATION TABLE OR FLOAT ARRAY ARRAY WHICH CONTAINS
X VALUES TO BE INTERPOLATED?NEWDATA<

COLUMN OF NEWDATA CONTAINING
X VALUES TO BE INTERPOLATED?1<

DO YOU WANT TO PERFORM EXTRAPOLATION FOR X VALUES
THAT MAY BE OUT OF THE RANGE OF X VALUES FROM THE SOURCE DATA?Y<

WARNING: X VALUE 1.1 IS OUT OF RANGE OF SOURCE DATA

INTERPOLATE COMPLETED. TOOK 1.704 CPU SEC.
```

Figure 3. Interacting with INTERPOLATE.

NEWDATA 9R X 2C

	1 NEW X	2 INTERPOLATED Y (ORIGDATA) ON NEW X
1	.325	3.064518
2	.645	4.1348
3	.865	5.27971
4	.234	2.887953
5	.111	2.744287
6	.976	5.988409
7	1.	6.14898
8	1.1	6.804286
9	.1	2.73449

Figure 4. New interpolated values.

PIV1A 8R X 3C

	1 SCHEDULE TIME	2 TIME ACTUAL	3 PLASMA CONC
1	0.	0.	10.
2	5.	4.	9.54851
3	10.	9.	9.012704
4	15.	14.	8.506964
5	30.	34.	6.752313
6	60.	55.	5.29803
7	120.	121.	2.472019
8	240.	237.	.6474169

PIV2A 8R X 3C

	1 SCHEDULE TIME	2 TIME ACTUAL	3 PLASMA CONC
1	0.	1.	9.885164
2	5.		
3	10.	13.	8.605789
4	15.	19.	8.029603
5	30.	36.	6.598122
6	60.	75.	4.205256
7	120.	100.	3.150575
8	240.	200.	.9926125

PIV3A 8R X 3C

	1 SCHEDULE TIME	2 TIME ACTUAL	3 PLASMA CONC
1	0.	0.	10.
2	5.	6.	9.330467
3	10.	8.	9.117404
4	15.	18.	8.122883
5	30.	27.	7.320913
6	60.	61.	4.94331
7	120.		
8	240.	270.	.4422345

Figure 5. A set of tables with X, Y data.

SUMTAB 8R X 1C

	1 SCHEDULE TIME
1	0.
2	5.
3	10.
4	15.
5	30.
6	60.
7	120.
8	240.

Figure 6. A destination table used to compute interpolation results for each set of X, Y data

```
>RAC PUBLIC INTERPOLATE<

>CALL MULTINTERPOLATE<
COLUMN OF SOURCE DATA TABLES WITH X VALUES?2<
COLUMN OF SOURCE DATA TABLES WITH Y VALUES?3<

WHEN PROMPTED, SUPPLY (ONE AT A TIME) THE NAMES
OF THE SOURCE DATA TABLES WHICH CONTAIN X DATA IN COLUMN 2
AND Y DATA IN COLUMN 3. WHEN FINISHED HIT <GO>.
SOURCE DATA TABLE NAME?PIV1A<
SOURCE DATA TABLE NAME?PIV2A<
SOURCE DATA TABLE NAME?PIV3A<
SOURCE DATA TABLE NAME?<

NAME OF DESTINATION TABLE WITH X VALUES TO BE INTERPOLATED?SUMTAB<
COLUMN NUMBER OF SUMTAB CONTAINING X VALUES?1<

DO YOU WANT TO PERFORM EXTRAPOLATION FOR X VALUES
THAT MAY BE OUT OF RANGE OF THE X VALUES FROM THE SOURCE TABLES?Y<
WARNING: X VALUE 240. IS OUT OF RANGE OF SOURCE DATA
WARNING: X VALUE 240. IS OUT OF RANGE OF SOURCE DATA
WARNING: X VALUE 0. IS OUT OF RANGE OF SOURCE DATA
```

Figure 7. Interacting with MULTINTERPOLATE.

SUMTAB 8R X 4C

	1 SCHEDULE TIME	2 INTERPOLATED PLASMA CONC (PIV1A) ON SCHEDULE TIME	3 INTERPOLATED PLASMA CONC (PIV2A) ON SCHEDULE TIME	4 INTERPOLATED PLASMA CONC (PIV3A) ON SCHEDULE TIME
1	0.	10.	9.994949	10.
2	5.	9.43854	9.447358	9.439383
3	10.	8.9093	8.913065	8.909403
4	15.	8.409213	8.408387	8.409454
5	30.	7.071392	7.072369	7.072126
6	60.	4.999727	4.999631	5.001779
7	120.	2.500406	2.507672	2.339098
8	240.	.6189644	.3759564	.5496164

Figure 8. Pooled interpolation results.

SMOOTH

by Jean Risley
Bolt Beranek and Newman Inc.

Summary

SMOOTH uses MLAB to remove some of the effects of noise from experimental measurements by smoothing the data with a hyperbolic curve.

Calling Sequence

```
RAC PUBLIC SMOOTH <GO>
CALL SMOOTH(data-table, X-col, Y-col)
           <table>   <fixed><fixed>
```

Citation

No citation necessary.

Questions or Comments

Contact Margaret Reid-Miller on the PROPHET line.

The public procedure SMOOTH provides a means of smoothing noisy experimental data. The technique it uses is a five-point variable internal hyperbolic smoothing algorithm provided by the MLAB system.

Using SMOOTH

To call SMOOTH, give the command

```
CALL SMOOTH(data-table, X-col, Y-col)
```

where data-table is the name of the table containing the noisy experimental data, and X-col and Y-col are the numbers of the columns containing the respective data values.

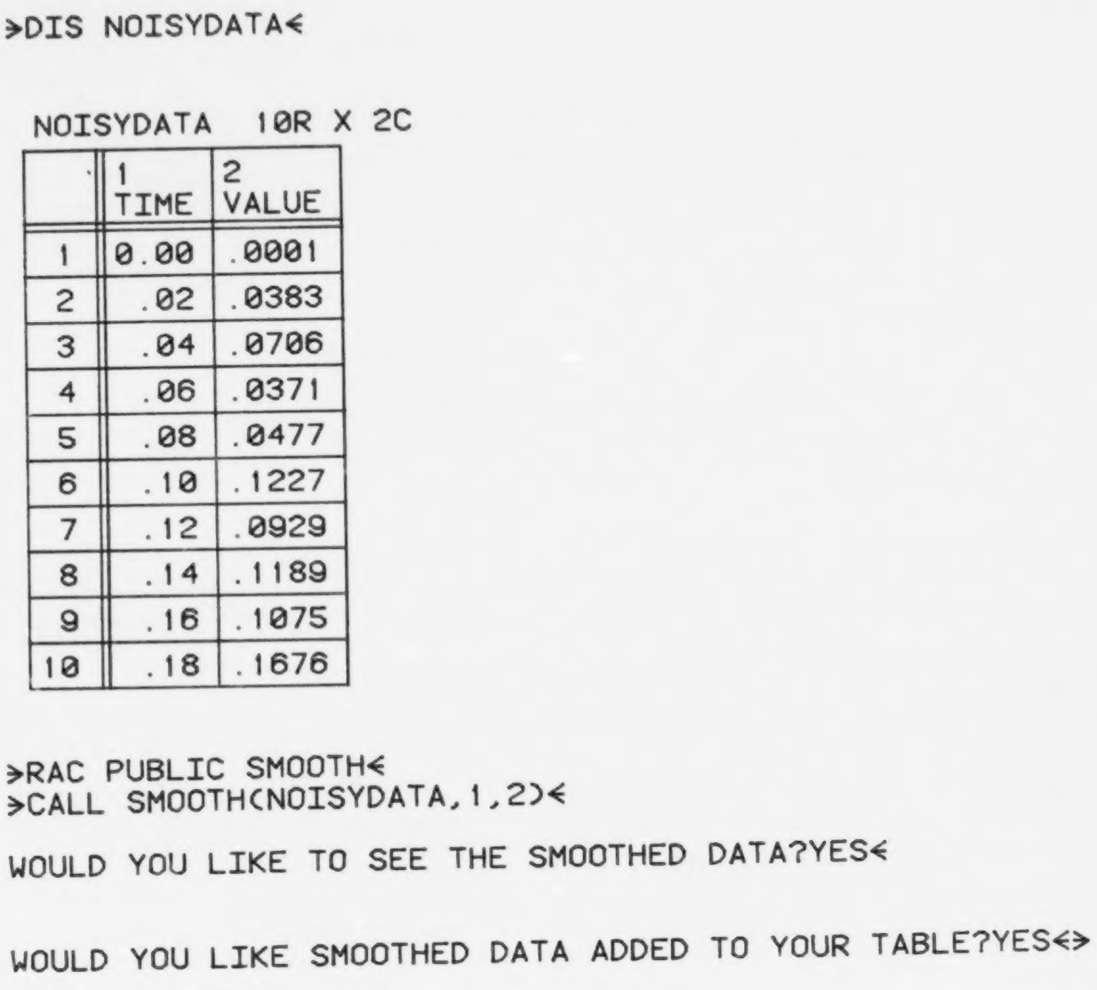
Once called, the procedure performs its computations and then asks whether you would like to see the results. If you answer YES, SMOOTH displays X-col and Y-col (sorted by ascending X values), along with a column of smoothed Y values. SMOOTH gives you the option of adding this column of smoothed values to data-table.

Errors

If you receive an error message from MLAB (e.g., BAD SYNTAX), type:

```
EXIT<GO>
```

This command returns you to top-level PROPHET, and you can try the procedure again.



```
>DIS NOISYDATA<
```

NOISYDATA 10R X 2C

	1 TIME	2 VALUE
1	0.00	.0001
2	.02	.0383
3	.04	.0706
4	.06	.0371
5	.08	.0477
6	.10	.1227
7	.12	.0929
8	.14	.1189
9	.16	.1075
10	.18	.1676

```
>RAC PUBLIC SMOOTH<
>CALL SMOOTH(NOISYDATA,1,2)<

WOULD YOU LIKE TO SEE THE SMOOTHED DATA?YES<


WOULD YOU LIKE SMOOTHED DATA ADDED TO YOUR TABLE?YES<>
```

Figure 1. Using SMOOTH.

10R X 3C

	1 TIME	2 VALUE	3 SMOOTHED VALUE
1	0.00	.0001	.01066388
2	.02	.0383	.03051575
3	.04	.0706	.04717583
4	.06	.0371	.05572396
5	.08	.0477	.06780848
6	.10	.1227	.09256558
7	.12	.0929	.09949449
8	.14	.1189	.118346
9	.16	.1075	.1286635
10	.18	.1676	.1560826

Figure 2. Smoothed values.

PLANIM

by Hank Baig
Bolt Beranek and Newman Inc.

Summary

PLANIM calculates the area within a closed curve (i.e., planimeters it).

Calling Sequence

```
RAC PUBLIC PLANIM <GO>
float-var = PLANIM
```

Entry

```
float-var = PLANIMQT(graf,  curve-num)
                    <graph>  <fixed>
```

Citation

No citation necessary.

Questions or Comments

Contact Hank Baig on the PROPHET line.

Things to Remember

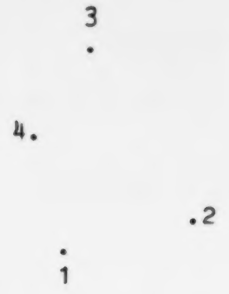
You must have entered your data into the graph in a counterclockwise direction.

PLANIM assumes that the last data point entered is connected to the first.

The public procedure PLANIM calculates the area within a closed curve.

Restrictions Two restrictions apply to the organization of the curve's points. You must have entered them into the graph in a counterclockwise direction, either from a table or by using the public procedure DIGITIZE. Also, PLANIM assumes that the last point you entered is connected to the first.

The counterclockwise sense is illustrated by the trivial case below, with the numbers representing the order of the data points.



Using PLANIM

To use PLANIM, issue the command:

```
float-var = PLANIM
```

The procedure prompts you for the name of the graph containing the curve you want to planimeter. If there are two or more curves in the graph, PLANIM asks you for the number of the one in which you are interested. It then returns the area bounded by that curve in the FLOAT variable, float-var.

Quiet Entry PLANIMQT

There is also a quiet entry to the procedure, PLANIMQT. You can call it with the command

```
float-var = PLANIMQT(graf, curve-num)
```

where graf is the name of a graph and curve-num is the index of a curve in the graph. If you leave out the second argument, PLANIMQT calculates the area bounded by curve number 1 in graf.

INTEGRAL

by Karl Kuhlmann
Stanford Research Institute

Summary

INTEGRAL calculates and returns the definite integral of a closed or nonclosed curve.

Calling Sequence

```
RAC PUBLIC INTEGRAL <GO>
```

Entries

For nonclosed curves:

```
(Graph  float- = INTEGRAL(curve-num,  graf,   low-X,  high-Y, base-line)
 Data)   var              <fixed>  <graph> <float> <float>   <float>


(Table  float- = INTEGRALTAB(tab,   X-col,  Y-col,  low-X,  high-X,
 Data)   var              <table> <fixed> <fixed> <float>  <float>

                           base-line)
                            <float>

(Array  float- = INTEGRALRA( Xs,    Ys,    low-X,  high-X, base-line)
 Data)   var              <float <float  <float>  <float>  <float>
                           array> array>
```

For closed curves:

```
(Graph  float-var = AREA(curve-num,  graf)
 Data)                    <fixed>   <graph>

(Table  float-var = AREATAB( tab,   X-col,  Y-col)
 Data)                     <fixed> <fixed> <fixed>

(Array  float-var = AREARA(     Xs,           Ys      )
 Data)                     <float array> <float array>
```

Citation

Kuhlmann K. INTEGRAL, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Send a message to 'KUHLMANN' using PROPHET.

Things to Remember

This procedure is particularly useful for curves entered using DIGITIZE, because INTEGRAL does not require that X values be monotonically increasing.

The approximation improves as the number of X values increases.

This procedure computes the area bounded by a curve. The curve may be closed (e.g., a circle) or not closed. If it is not closed, you must provide both a baseline and limits for X values. INTEGRAL calculates the area by using a trapezoidal approximation to a line integral. This approximation improves with the number of data points in the curve.

The procedure offers three entries for closed curves and three entries for nonclosed curves. Different entries are provided for data based in a graph, in a pair of arrays, and in a table.

Entries for Nonclosed Curves

Use the following entries to compute the area under a non-closed curve. (These entries behave identically except for the way you specify the X,Y data.)

```
(1) float-var = INTEGRAL(curve-num, graf, low-X,
                         high-X, base-line)
```

Curve curve-num of the graph graf is used as the curve data.

```
(2) float-var = INTEGRALTAB(tab,  X-col, Y-col,
                        low-X, high-X, base-line)
```

Columns X-col and Y-col of the table tab are used as the curve data.

(3) float-var = INTEGRALRA (Xs, Ys, low-X, high-X, base-line)

Arrays Xs and Ys are used as the curve data.

Upon return from each of these entries, the FLOAT variable int contains the area under the specified curve for X values between low-X and high-X using the baseline specified by base-line. If high-X < low-X, the integral is multiplied by -1. If either low-X or high-X is empty, the procedure automatically uses the minimal or maximal X value, respectively.

Entries for Closed Curves

Use the following entries to compute the area bounded by a closed curve. You need not specify X limits or a baseline. Different entries are available for curves specified by a graph, by a table, and by a pair of arrays.

(1) float-var = AREA(curve-num, graf)

Curve curve-num of the graph graf is used as the curve data.

(2) float-var = AREATAB(tab, X-col, Y-col)

Columns X-col and Y-col of the table tab are used as the curve data.

(3) float-var = AREARA(Xs, Ys)

Arrays Xs and Ys are used as the curve data.

TRAP

by Jean Risley
Bolt Beranek and Newman Inc.

Summary

TRAP computes the area under a curve using the trapezoidal rule.

Calling Sequence

```
RAC PUBLIC TRAP <GO>
float-var = TRAP(data-tab)
                  <table>
```

Entry

```
float-var = TRAPQT(data-tab, X-col,  Y-col, sum-type)
                   <table>  <fixed> <fixed>  <text>
```

Citation

No citation necessary.

Questions or Comments

Contact Margaret Reid-Miller on the PROPHET line.

Things to Remember

The X values in X-col must be in ascending order.

TRAP's Computations

The public procedure TRAP uses the trapezoidal rule to compute the approximate area under a curve. After inputting a series of sample points (X,Y) from the columns of a table, the procedure calculates an approximate area as follows:

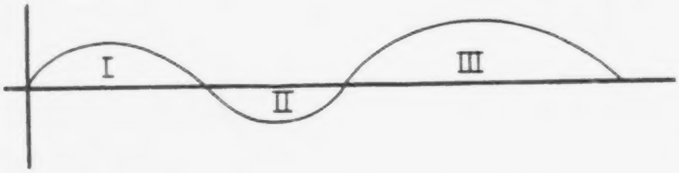
$$Y = f(X)$$

$$\int_{X_i}^{X_{i+1}} f(X)\,dX \approx (X_{i+1} - X_i) \cdot \left[\frac{f(X_{i+1}) + f(X_i)}{2} \right] = A_i$$

$$S_{abs} = \sum_{i=1}^{N} |A_i|$$

$$S_{sum} = \sum_{i=1}^{N} A_i$$

The sum of areas is the sum of the areas under straight lines joining the input data points. Note that the more closely spaced your data points, the better the approximation to the true area. The two forms of the sum -- S_{abs} and S_{sum} -- are different only when the function crosses the X axis. For example, for the function



the absolute sum (S_{abs}) is the sum of areas I, II, and III, while the straight sum (S_{sum}) is area I plus area III minus area II.

Using TRAP Interactively

To use TRAP, type the command:

```
float-var = TRAP(data-tab)
```

TRAP asks you for the numbers of the columns in data-tab that contain the X and Y values. It also asks whether you want the absolute sum or the straight sum computed. Then it returns the sum in the FLOAT variable, float-var.

Note that the X values must be in ascending order. If they are out of order, use the SORT command (see Section 3 of the <u>PROPHET User's Manual</u>) to sort them into ascending order before calling TRAP.

Quiet Entry TRAPQT

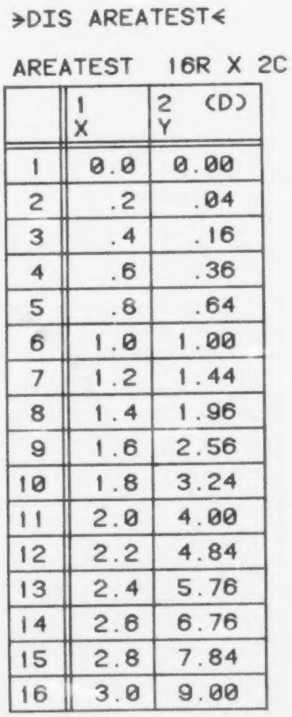
There is also a quiet entry TRAPQT. You can call it by typing the command

```
float-var = TRAPQT(data-tab, X-col, Y-col,
                   sum-type)
```

where data-tab is the table containing the data; X-col is the FIXED number of the column containing the X values; Y-col is the FIXED number of the column containing the Y values; and sum-type is a TEXT that must be set either to 'ABS' (absolute sum) or to 'SUM' (straight sum). The sum returns in the FLOAT variable, float-var.

Example

In the following example, TRAP calculates the area under the curve $Y = X^2$ on the interval [0.,3.] with 14 intermediate values.



```
>DIS AREATEST<
```

AREATEST 16R X 2C

	1 X	2 (D) Y
1	0.0	0.00
2	.2	.04
3	.4	.16
4	.6	.36
5	.8	.64
6	1.0	1.00
7	1.2	1.44
8	1.4	1.96
9	1.6	2.56
10	1.8	3.24
11	2.0	4.00
12	2.2	4.84
13	2.4	5.76
14	2.6	6.76
15	2.8	7.84
16	3.0	9.00

Figure 1. A typical data table for TRAP.

```
>RAC PUBLIC TRAP<
>T=TRAP(AREATEST)<

COLUMN NUMBER OF X-VALUES = ?1<
COLUMN NUMBER OF Y-VALUES = ?2<   COMPUTE ABSOLUTE SUM?Y

>TYPE T< 9.02>
```

Figure 2. Using TRAP.

STEPAREA

by Jean Risley
Bolt Beranek and Newman Inc.

Summary

STEPAREA uses a cubic spline interpolation to calculate and return the area under an exponential decay curve.

Calling Sequence

```
RAC PUBLIC STEPAREA <GO>
CALL STEPAREA(data-tab)
              <table>
```

or

```
float-var = STEPAREA(data-tab)
                     <table>
```

Entry

```
float-var = STEPAREAQT(data-tab, X-col,  Y-col, extrap-flag,
                       <table> <fixed> <fixed>   <bool>

                       add-col-flag)
                          <bool>
```

Citation

No citation necessary.

Questions or Comments

Contact Margaret Reid-Miller on the PROPHET line.

Things to Remember

The X values in X-col must be in ascending order.

STEPAREA's Computations

STEPAREA computes the approximate area under a curve using a cubic spline interpolation method. You supply a series of sample points (X,Y) in the columns of a table. For each pair of points, the procedure calculates an approximate area by first using interpolation to divide the interval into eight equally spaced subintervals and then summing the areas of the subintervals. STEPAREA assumes that the functional form is an exponential decay, and its correction of the area of the last interval is based on this assumption. (The procedure's output also includes the result obtained using uncorrected area for the last interval.)

Interacting with STEPAREA

To use STEPAREA, type the command:

```
CALL STEPAREA(data-tab)
```

The procedure asks you for the numbers of the columns in data-tab that contain the X and Y values. It also asks whether you want the area extrapolated to the asymptote, and whether you would like a column of incremental areas added to data-tab.

Note that the X values must be in ascending order. If they are out of order, use the SORT command (see Section 3 of the <u>User's Manual</u>) to sort them into ascending order before calling STEPAREA.

Quiet Entry, STEPAREAQT

STEPAREA also has a quiet entry, STEPAREAQT. You can call the quiet entry by typing the command:

```
float-var = STEPAREAQT(data-tab, X-col, Y-col,
                       extrap-flag,add-col-flag)
```

where data-tab is the table containing the data; X-col is the FIXED number of the column containing the X values; Y-col is the FIXED number of the column containing the Y values; extrap-flag is a BOOL that should be TRUE if you want to extrapolate to the asymptote; and add-col-flag is a BOOL that should be TRUE if you want to add the column of areas to your table. The uncorrected total area returns in the FLOAT variable, float-var.

Output

The output of STEPAREA is a display of the incremental area under the curve at the end of each interval and the uncorrected total area at the end of the last interval. If you choose to extrapolate to the asymptote, the procedure asks you to provide the slope for extrapolation and then returns the total area to the asymptote.

See Figure 1 for details.

```
>RAC PUBLIC STEPAREA<
>CALL STEPAREA(AREATEST)<

COLUMN NUMBER OF X VALUES = ?1<
COLUMN NUMBER OF Y VALUES = ?2<
DO YOU WANT AREA EXTRAPOLATED TO ASYMPTOTE ?NO<
WOULD YOU LIKE A COLUMN OF INCREMENTAL AREAS ADDED TO YOUR TABLE?YES<

TOTAL AREA =9.

UNCORRECTED AREA TO LAST DATA POINT = 9.

1 COLUMN ADDED
>
```

AREATEST 16R X 3C

	1 X	2 (D) Y	3 INCREMENTAL AREAS
1	0.0	0.00	0.
2	.2	.04	.002666667
3	.4	.16	.02133333
4	.6	.36	.072
5	.8	.64	.1706667
6	1.0	1.00	.3333333
7	1.2	1.44	.576
8	1.4	1.96	.9146667
9	1.6	2.56	1.365333
10	1.8	3.24	1.944
11	2.0	4.00	2.666667
12	2.2	4.84	3.549333
13	2.4	5.76	4.608
14	2.6	6.76	5.858667
15	2.8	7.84	7.317333
16	3.0	9.00	9.

Figure 1. Using STEPAREA.

4. Modelling Procedures

4. Modelling Procedures

4. Modelling Procedures

PROBIT performs probit analysis of dose-response data for estimating ED50 4-1

AGONIST analyzes dose-response curves to estimate full or partial agonist dissociation constant . 4-7

ANTAGONIST analyzes dose-response curves to estimate competitive antagonist dissociation constant . 4-17

LOGISTIC performs iterative least-squares fit of a logistic function 4-27

ABSORB computes and plots absorption profiles . . . 4-35

DIFFEQ fits and integrates systems of differential equations. 4-43

DRUGFUN fits parameters of standard pharmacokinetic models. 4-63

DRUGAUC uses DRUGFUN results to estimate the area under a concentration-time curve plus related measures. 4-81

MKMODEL performs simulation or curve fitting of one or more functions 4-89

GOFIT displays goodness-of-fit measures for models fitted with DRUGFUN or MKMODEL. 4-127

GRAFIT graphically displays the results of DRUGFUN or MKMODEL 4-131

MODELTEST compares models used to fit curves to a common set of observations 4-139

CELLSIM simulates patterns of growth in cell populations 4-147

PROBIT

by Jean Risley
Bolt Beranek and Newman Inc.

Summary

PROBIT transforms quantal dose-response data into probits and fits a straight line to the probit values.

Calling Sequence

```
     RAC PUBLIC PROBIT <GO>
     CALL PROBIT(data-table)
                 <table>

or

     function-text = PROBIT(data-table)
                            <table>
```

Citation

Risley, J. PROBIT, in *PROPHET Public Procedures Notebook* (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Contact Margaret Reid-Miller on the PROPHET line.

Things to Remember

The algorithm computing the threshold response is very sensitive and may not converge for mildly noisy data.

The column of data-table used in PROBIT must contain no empty cells.

The public procedure PROBIT performs probit analysis of dose-response data. You input a table containing dose, response, and sample size data, and the procedure outputs the threshold rate and the intercept and slope of the probit line. The technique used is that provided in the BMD package of biomedical computer programs by the Health Sciences Computing Facility, University of California at Los Angeles.

Calling the Procedure

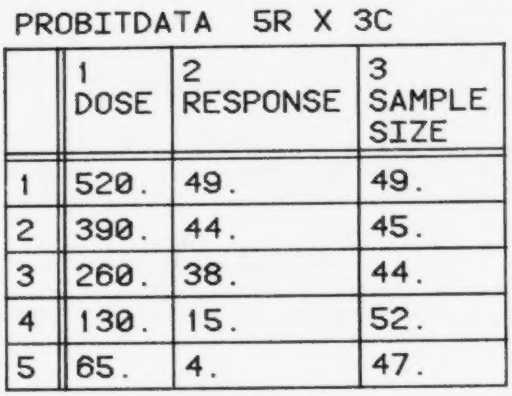
To use the procedure, type the command:

```
CALL PROBIT(data-table)
```

where data-table is the table containing the dose-response data. If you want the text of the fitted linear equation, use PROBIT as a function:

```
function-text = PROBIT(data-table)
```

The TEXT of the equation returns in function-text.



PROBITDATA 5R X 3C

	1 DOSE	2 RESPONSE	3 SAMPLE SIZE
1	520.	49.	49.
2	390.	44.	45.
3	260.	38.	44.
4	130.	15.	52.
5	65.	4.	47.

Figure 1. A typical data table for PROBIT.

Interacting with PROBIT

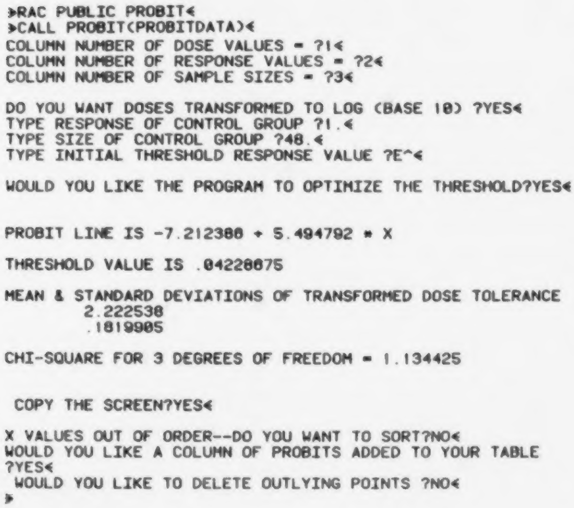
PROBIT first asks you to provide the numbers of the columns of your data table that contain the dose, response, and sample size data.

Then, after asking whether you would like to perform a logarithmic transformation of the dose values, PROBIT prompts you for:

- the response number of the control group
- the size of the control group
- an initial estimate of the threshold values

Whether or not you supply these data is optional. If you cannot or do not want to provide a particular value, type CTRL-E in response to the procedure's prompt. If you do make an initial threshold guess, PROBIT also asks you to estimate the probit line's intercept and slope. Again, answering these questions is optional.

Finally, before computing the probit line, PROBIT asks you whether you would like the procedure to maximize the threshold value (C) or hold it constant.



```
▸RAC PUBLIC PROBIT◂
▸CALL PROBIT(PROBITDATA)◂
COLUMN NUMBER OF DOSE VALUES = ?1◂
COLUMN NUMBER OF RESPONSE VALUES = ?2◂
COLUMN NUMBER OF SAMPLE SIZES = ?3◂

DO YOU WANT DOSES TRANSFORMED TO LOG (BASE 10) ?YES◂
TYPE RESPONSE OF CONTROL GROUP ?1.◂
TYPE SIZE OF CONTROL GROUP ?48.◂
TYPE INITIAL THRESHOLD RESPONSE VALUE ?E^◂

WOULD YOU LIKE THE PROGRAM TO OPTIMIZE THE THRESHOLD?YES◂


PROBIT LINE IS -7.212380 + 5.494792 * X

THRESHOLD VALUE IS .04228075

MEAN & STANDARD DEVIATIONS OF TRANSFORMED DOSE TOLERANCE
        2.222538
         .1819905

CHI-SQUARE FOR 3 DEGREES OF FREEDOM = 1.134425


 COPY THE SCREEN?YES◂

X VALUES OUT OF ORDER--DO YOU WANT TO SORT?NO◂
WOULD YOU LIKE A COLUMN OF PROBITS ADDED TO YOUR TABLE
?YES◂
 WOULD YOU LIKE TO DELETE OUTLYING POINTS ?NO◂
▸
```

Figure 2. Interacting with PROBIT.

After PROBIT calculates the probit line and associated statistics (described below under "PROBIT's Computations"), it displays a graph of the actual and computed probits versus the input doses. You now have the options of adding a column of probits to your data table and of rerunning the entire probit analysis, ignoring any number of specified outlying points.

If your input data are not sorted in order of increasing dose, PROBIT now asks whether you would like to sort them. If you want to rerun the probit analysis, you must first sort your data and then answer YES to the question:

WOULD YOU LIKE TO DELETE OUTLYING POINTS?

If you do answer YES, PROBIT redisplays the graph with each of the data points identified by a number. You then type in the list of points to ignore by typing the numbers each followed by <GO>. After you enter the last point number, type 0 to indicate the end of the list. PROBIT performs a probit analysis on the remaining points and displays a new graph. If you like, you can then repeat this process, ignoring additional points.

Note that the outlying points you ignore are excluded only from the analysis; they are not deleted from the data table.

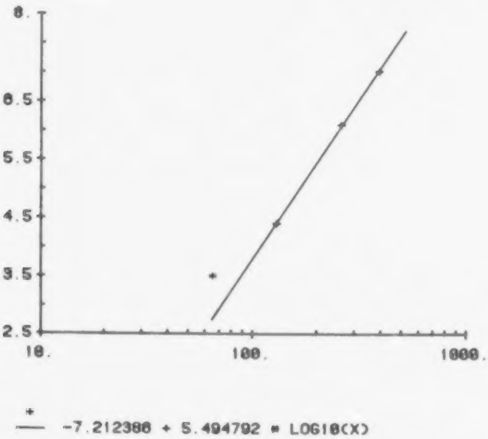


Figure 3. Fitted probit equation, associated statistics, and plot of actual and computed probits versus input doses.

PROBIT's Computations

The computations performed by PROBIT are detailed below.

Definitions:

Sample probability

$$p_i = \frac{(-C_0 + r_i/n_i)}{1 - C_0}$$

Sample probit

$$Y_i = \Phi^{-1}(p_i) + 5$$

where Φ is the cumulative standard normal distribution function.

Given r_i, the response of a sample of size n_i to a dose d_i for $i = 1, \ldots, M$, PROBIT computes a specified transformation on the dose:

$$T: d_i \to X_i \qquad i = 1, \ldots, M$$

The starting values α_0 and β_0 of α and β are obtained by linear regression of the sample probit as a function of the tranformed dose. For this step, PROBIT ignores the control group, as well as those doses corresponding to a sample probability of zero or one.

PROBIT obtains the starting value, C_0, as follows:

C_0 = an inputted value, if you supply one
or
C_0 = (control-group response)/(control-group size)
if you supply these data and they do not
equal zero
or
C_0 = min (r_i/n_i) if all $r_i > 0$
= $1/(2n_i)$ if $r_i = 0$ for some i

The procedure improves the initial values of α and β by following the interative scheme but holding C constant. When it has obtained convergence for improved α and β, it then follows the whole iterative scheme for all three parameters if you required that C be optimized.

As an aid to convergence, PROBIT carries out the iterative scheme with X_i replaced by $X_i - \bar{X}$. In the final stage, the procedure makes compensating adjustments to give results in terms of the original dose. (For a description of the iterative scheme, consult the references listed below.) The values μ and σ are computed by means of the equations

$$\mu = (5 - \alpha)/\beta$$

$$\sigma = {}^{1}/\beta$$

The Chi-squared statistic is computed and printed along with the degrees of freedom (d.f.):

$$X^2 = \sum_{i=1}^{M} \frac{(r_i - n_i p_i)^2}{n_i p_i (1 - p_i)}$$

d.f. = M - 2 when C is constant
M - 3 when C is optimized without control group
M - 2 when C is optimized with control group

References

Finney DJ. Probit Analysis, 2nd ed. Cambridge Univeristy Press, 1952.

Dixon WJ and Massey FJ. Introduction to Statistical Analysis, 3rd ed. McGraw-Hill, 1969. Section 15.2.

AGONIST

by Carl Johnson
Mt. Sinai School of Medicine

Summary

AGONIST evaluates the dissociation constant of an agonist by comparing the dose-response curves before and after partial receptor blockage with an irreversible antagonist. It further evaluates the dissociation constant of a partial agonist by comparing the dose-response curves of the partial agonist and a standard full agonist.

Calling Sequence

```
RAC PUBLIC AGONIST <GO>
CALL AGONIST(control-table, exp-table)
                <table>       <table>
```

Entry

```
CALL AGONIST2(control-table, exp-table, data-plot)
                 <table>       <table>    <graph>
```

Citation

Johnson C. AGONIST, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions and Comments

Contact Jim Wood on the PROPHET line.

Things to Remember

The algorithm AGONIST uses is suitable only for curves having the shape shown in Figure 4.

Analytical Background and Methods

The public procedure AGONIST is designed to carry out either of the following analyses of pharmacological data:

(1) Evaluation of the dissociation constant of an agonist by comparing the dose-response curve before and after partial receptor blockade with an irreversible antagonist.

(2) Evaluation of the dissociation constant of a partial agonist by comparing the dose-response curves of the partial agonist with those of a standard full agonist.

The procedure is a modification of the method suggested by Waud and Parker and is based on the assumptions that agonists react reversibly with the receptor according to the laws of mass action, and that the pharmacological response is some (unknown) function of the receptor occupation:

$$Y_A = \frac{[A]}{[A] + K_A}$$

$$E_A = f(Y_A)$$

where Y_A = fractional receptor occupation by A
$[A]$ = agonist concentration
K_A = dissociation constant of agonist
E_A = pharmacological response
f = unknown function

Method 1

In Method 1, the effect of the irreversible blocker is to reduce the total number of available receptors so that

$$Y'_A = \frac{[A]'}{[A]' + K_A} \cdot (1 - Y_I)$$

$$E'_A = f(Y'_A)$$

where Y_I is the fraction of receptors blocked, and the primes indicate the presence of the blocking agent. If the responses in the presence and absence of the blocking agent are matched

$$E_A = E'_A$$

then a least-squares fit of the equation

$$\frac{[A]}{[A] + K_A} = \frac{[A]'}{[A]' + K_A} \cdot (1 - Y_I)$$

yields values for K_A and Y_I .

Method 2

In Method 2, the assumption is made that the function f can be split into two parts, a linear drug-dependent part (the efficacy) and a nonlinear drug-independent part, g :

$$E_A = f(Y_A) = g(\epsilon_A Y_A)$$

Matching the responses of a full and a partial agonist leads to

$$E_A = E_P$$

$$\epsilon_A \cdot \frac{[A]}{K_A} = \epsilon_P \cdot \frac{[P]}{[P] + K_P} \quad \text{, since } [A] << K_A$$

A least-squares fit of this equation yields values for K_P and $K_A \epsilon_P / \epsilon_A$ (the physical meaning of this last term is not clear).

AGONIST's Computations

The analysis is carried out in three steps:

Step 1

AGONIST fits the control and experimental curves using a logistic function of the form

$$E = M \frac{[A]^S}{[A]^S + L^S}$$

where M = maximal response,
L = ED_{50},

and the slope of the curve is determined by S. AGONIST fits the logistic function iteratively using a Taylor expansion in the nonlinear parameters, L and S. You must supply initial estimates of these parameters. We have found empirically that a visual estimate of the ED_{30} (in preference to the ED_{50}) provides a suitable initial estimate of L. A convenient initial estimate of the slope parameter, S, is twice the observed slope, because differentiating the logistic function leads to:

$$S = 1.74 \frac{d\ (E/M)}{d\,(\log[A])}$$

Grossly over- or underestimating either L or S may lead to negative values for L or S during the iteration. If this occurs, the procedure will start over.

Step 2

Using the fitted experimental curve, AGONIST calculates responses corresponding to the [A]' or [P] values.

With these calculated responses ($\hat{E}$), the procedure then determines new [A] values from the fitted control curve. That is, for every value of [A]' (or [P]), the procedure finds values of [A] such that the control curve and the experimental curve have the same response level. Using the following equation, the procedure then carries out a least-squares fit of the computed [A] versus [A]' (or [P]) data:

$$[A] = \frac{M[A]'}{[A]' + L}$$

Method 1:

$$M = K_A\,(1 - Y_I)/Y_I$$

$$L = K_A/Y_I$$

Method 2:

$$M = K_A\epsilon_P/\epsilon_A$$

$$L = K_P$$

The fit is again accomplished with an iterative procedure. The procedure automatically determines the initial estimate of L by means of a weighted fit of the linear form of the above equation (i.e., a double-reciprocal plot of 1/[A] versus 1/[A]').

Step 3

AGONIST uses final converged values of L and M to calculate K_A and Y_I (Method 1) or K_P and $K_A\epsilon_P/\epsilon_A$ (Method 2). Although error estimates for the parameters could be derived using standard statistical procedures, it is probably more practical to calculate the K_A's for several preparations and then treat the set of K_A's as a random sample in the usual way.

Calling the Procedure

To use AGONIST, type the command:

```
CALL AGONIST(control-table, exp-table)
```

where control-table is the name of the table containing the control drug data, and exp-table is the name of the table containing the experimental drug data.

Column 1 of control-table should contain the agonist concentration, and column 2 should contain the response data. Column 1 of exp-table must contain the agonist concentration after receptor blockade or the partial agonist concentration; column 2 must contain the response data. The procedure makes a semilog plot of these data and asks whether you want to examine it.

AGON1: CONTROL 7R X 2C

	1 NG 5-OH-T	2 EFFECT MM
1	.18	3.00
2	.44	3.50
3	.88	6.33
4	1.80	11.17
5	4.40	16.70
6	8.80	20.83
7	17.60	25.30

AGON2: EXPERIMENTAL 6R X 2C

	1 NG 5-OH-T	2 EFFECT MM
1	1.80	2.75
2	4.40	6.00
3	8.80	7.75
4	17.60	9.25
5	44.00	10.75
6	88.00	10.50

```
>RAC PUBLIC AGONIST<
>CALL AGONIST(AGON1,AGON2)<

DO YOU WANT TO EXAMINE A PLOT OF THE DATA?NO<
```

Figure 1. AGONIST input tables and initial interaction.

Quick Entry AGONIST2

If you have already made a semilog plot (log X axis) of the data in the two tables, you can use the entry

```
CALL AGONIST2(control-table, exp-table,
                   data-plot)
```

to save time.

Iterative Computation of L and S

AGONIST (or AGONIST2) next asks whether you want to use the automatic convergence feature. If you answer YES, it asks for the convergence value, expressed as a percentage. For example, if you choose a convergence value of 0.1, the iterations used to calculate the fitted parameters automatically continue until the current values of the parameters differ from the last values by less than 0.1% (i.e., by a factor of 0.001).

It may be impossible to achieve convergence if you choose a very small convergence value (e.g., 0.00001%). Consequently, the number of iterations is allowed to go only up to 20. If the procedure continues this far, the values of the parameters calculated at the 19th and 20th iterations are typed out and the procedure asks whether you want to use the last values and continue in AGONIST. If you do, answer YES.

AGONIST next asks you to supply an initial estimate of L and S for the control data. If you chose automatic convergence, iteration continues until an acceptable convergence is reached, and the procedure types out the converged values. Otherwise, AGONIST types out the values for L and S and continues the iterations only so long as you answer YES to the question

```
CONTINUE?
```

If the procedure computes a negative or zero value for L or S (because of bad initial estimates), iteration stops and you are asked to supply new estimates.

```
DO YOU WANT AUTOMATIC CONVERGENCE?NO<
              CONTROL CURVE

     INITIAL ESTIMATE OF L = 4.<
INITIAL ESTIMATE OF S(LOPE) = .8<

L = 4.421955          S = .8330645 CONTINUE? Y<
L = 4.35021           S = .8409016 CONTINUE? Y<
L = 4.344366          S = .8414641 CONTINUE? Y<
L = 4.343812          S = .8415065 CONTINUE? Y<
L = 4.343768          S = .8415098 CONTINUE? Y<
L = 4.343761          S = .8415103 CONTINUE? NO<

THE CURVE IS FITTED BY THE EQUATION:
E = 32.90517[A]**.8415103/([A]**.8415103
    + 4.343761**.8415103)
```

Figure 2. Iterative computation of L and S for the control data.

Once the control data have converged, AGONIST asks you to supply estimates of L and S for the experimental data. Iterations are carried out as described for the control data. AGONIST uses the iterative process to produce convergence for the parameter L.

```
              EXPERIMENTAL CURVE

     INITIAL ESTIMATE OF L = 4.<
INITIAL ESTIMATE OF S(LOPE) = 2.25<

L = 3.814048          S = .9806886 CONTINUE? Y<
L = 4.00108           S = 1.269702 CONTINUE? Y<
L = 4.056904          S = 1.25665  CONTINUE? Y<
L = 4.059256          S = 1.256386 CONTINUE? Y<
L = 4.059298          S = 1.256365 CONTINUE? Y<
L = 4.0593            S = 1.256365 CONTINUE? NO<

THE CURVE IS FITTED BY THE EQUATION:
E = 10.91004[A]**1.256365/([A]**1.256365
    + 4.0593**1.256365)


S = 8.724266     CONTINUE? Y<
S = 9.500008     CONTINUE? Y<
S = 9.46057      CONTINUE? Y<
S = 9.465532     CONTINUE? Y<
S = 9.464925     CONTINUE? Y<
S = 9.465        CONTINUE? Y<
S = 9.46499      CONTINUE? NO<
```

Figure 3. Iterative computation of L and S for the experimental data.

Irreversible and Partial Agonists

AGONIST then asks whether you are using Method 1 (irreversible antagonist) or Method 2 (partial agonist). Answer by typing 1 or 2. The procedure types out final values, adds the fitted curves to the graph, and prints them. To obtain a top-level copy of the graph, supply a name for it in response to AGONIST's prompt.

```
METHOD (IRREVERSIBLE ANTAGONIST = 1; PARTIAL AGONIST = 2) ? 1<

FINAL ESTIMATE OF K(AGONIST) = 7.352786

RECEPTOR OCCLUSION = 77.68403
%
```

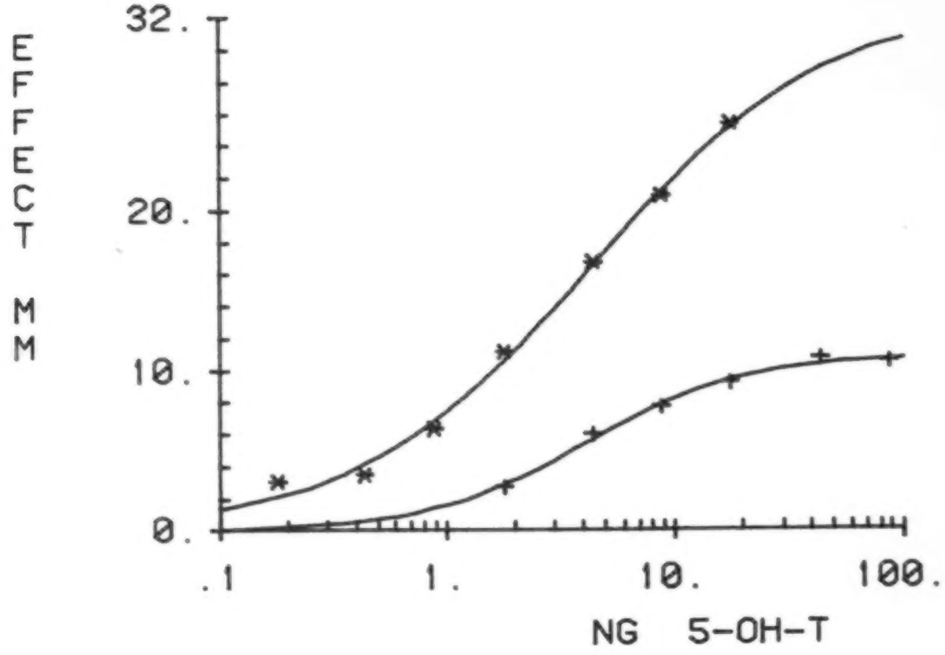


Figure 4. AGONIST's final output.

Notes

- A poor choice of initial estimates may lead to divergence rather than convergence of L and S during the iterative fit of the logistic functions. When AGONIST asks whether it should continue, type RESTART to begin at the top or STOP to exit from the procedure.

- The logistic fit is unweighted because the error in the pharmacological response is usually not a function of the magnitude of the response. (You should verify this for your assay system.)

- The [A] versus [A]' fit is weighted based on the observation that in a series of computer-simulated experiments, the error in [A] was proportional to the magnitude of [A] (see the reference cited below).

- We have sometimes experienced difficulty fitting the experimental curve because of scattering of the response data. This is particularly a problem with the end points of the dose-response curve (at low and high agonist concentrations), which can markedly affect the shape of the fitted curve. On these occasions, we have eliminated points that deviated markedly from the sigmoidal dose-response curve tc obtain a converged fit.

- The original method of Waud and Parker did not fit the experimental responses when used in Step 2 to calculate the set of [A] values. Because considerable variation in the observed responses is possible, we believe it is preferable to take the experimental responses off a smoothly fitted curve.

Reference

Waud DR and Parker RB. Pharmacological estimation of drug-receptor dissociation constants. Statistical evaluation. I. Agonist. J Pharmacol Exp Ther 177:1-12, 1971.

ANTAGONIST

by Carl Johnson
Mt. Sinai School of Medicine

Summary

ANTAGONIST calculates the dissociation constant of a competitive antagonist by measuring the shift in the dose-response curve of an agonist caused by the presence of the antagonist.

Calling Sequence

```
RAC PUBLIC ANTAGONIST <GO>
CALL ANTAGONIST(drug-table)
                <table>
```

Entry

```
CALL ANTAGONIST2(drug-table, data-plot)
                 <table>     <graph>
```

Citation

Johnson, C. ANTAGONIST, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Contact Jim Wood on the PROPHET line.

Things to Remember

Drug-table must be organized so that column 1 contains the agonist concentration, column 2 contains the control responses, and the remaining columns contain experimental responses.

The algorithm used by ANTAGONIST is suitable only for curves having the shape shown in Figure 2.

Analytical Background and Methods

The public procedure ANTAGONIST calculates the dissociation constant of a competitive antagonist by comparing the dose-response curves of an agonist in the absence of the antagonist and in the presence of variable concentrations of the antagonist. The procedure follows the method suggested by Waud and Parker, and is based on the assumptions that an antagonist reacts with the receptor according to the laws of mass action and that the observed pharmacological response is some (unknown) function of the degree of receptor occupation,

$$Y = \frac{[A]}{[A] + K_A}$$

$$E = f(Y)$$

where Y = fraction of receptors occupied
$[A]$ = agonist concentration
K_A = agonist dissociation constant
E = pharmacological response
f = unknown function

By definition, a competitive antagonist shifts the agonist dose-response curve to higher agonist concentration without affecting either the maximal effect or the slope of the curve:

$$Y' = \frac{[A]'}{[A]' + K_A(1 + [B]/K_B)}$$

$$E' = f(Y')$$

where $[B]$ = antagonist concentration
K_B = antagonist dissociation constant

and primes indicate the presence of antagonist. Equating responses in the absence and presence of antagonist

$$E = E'$$

leads to the "dose ratio" equation

$$\text{dose ratio} - 1 = \frac{[A]'}{[A]} - 1 = \frac{[B]}{K_B}$$

or

$$\log(DR-1) = -\log(K_B) + \log[B]$$

A least-squares fit of $\log(DR-1)$ versus $\log[B]$ yields the antagonist dissociation constant, K_B .

Calling ANTAGONIST

To use the procedure, type

```
CALL ANTAGONIST(drug-table)
```

where drug-table is the name of a table containing dose-response data arranged as follows:

COL 1 = agonist concentrations
COL 2 = control responses (absence of antagonist)
COL 3 to N = experimental responses (presence of variable concentrations of antagonist)

ANTAGONIST makes a semilog plot of these data and asks whether you want to examine it (Figure 1).

```
>DIS ANTAG<
```

ANTAG: ANTAGONIST DRUG DATA 8R X 5C

	1 [A] (X E+7 M)	2 CONTROL	3 D-TC (1.E-U M)	4 D-TC (2.E-7 M)	5 D-TC (4.E-7 M)
1	2.	.5			
2	4.	2.2			
3	5.		.7		
4	8.	5.5	1.8		
5	10.			1.1	
6	15.	8.0	5.3	2.7	.8
7	25.			4.8	3.0
8	40.				4.9

```
>RAC PUBLIC ANTAGONIST<
>CALL ANTAGONIST(ANTAG)<

DO YOU WANT TO EXAMINE A GRAPH OF THE DATA?NO<
```

Figure 1. ANTAGONIST drug table and initial interaction.

Quick Entry ANTAGONIST2

If you have already made a semilog plot (log X axis) of your drug data, you can save time by using the command

```
CALL ANTAGONIST2(drug-table, drug-plot)
```

where drug-plot is the name of your graph.

The analysis is carried out in four steps:

Common Slope Fit

Step 1. ANTAGONIST fits all of the dose-response curves using logistic functions with common slope:

$$E_i = \frac{M[A]_i^S}{[A]_i^S + L_i^S}$$

where M = maximal effect
S = slope parameter
i = curve label
L_i = ED_{50} for curve i

The procedure fits the logistic functions iteratively using a Taylor expansion in the nonlinear parameters and S; you supply initial estimates. An automatic convergence option is available to simplify the iteration. You should provide a convergence factor (expressed as a percentage). If you supply a convergence factor of 0.1, iteration continues until all the present values of L and S differ from the last values by less than 0.1%.

We have found that a visual estimate of the ED_{30} (in preference to the ED_{50}) provides a suitable initial estimate of the L_i. A convenient initial estimate of the slope parameter S is twice the observed slope, because differentiating the logistic function leads to:

$$S = 1.74\,\frac{d(E/M)}{d(\log[A])}$$

Grossly over- or underestimating either the L_i or S may lead to negative values for L_i or S during the iteration. If this occurs, the procedure starts over. ANTAGONIST continues iteration until it obtains convergence in the values of L_1 and S. It then displays and prints a graph of the fitted dose-response curves (Figure 2).

```
DO YOU WISH TO USE AUTOMATIC CONVERGENCE?YES<
  ENTER CONVERGENCE FACTOR (IN PERCENT) = ?.1<

                    LOGISTIC FIT -- COMMON SLOPE

INITIAL ESTIMATE OF COMMON S(LOPE) =  2.<

INITIAL ESTIMATE OF L, CURVE 1 =  5.<
INITIAL ESTIMATE OF L, CURVE 2 =  10.<
INITIAL ESTIMATE OF L, CURVE 3 =  15.<
INITIAL ESTIMATE OF L, CURVE 4 =  20.<

               CONVERGED VALUES:

CURVE 1 :  L = 6.593639    S = 2.353057
CURVE 2 :  L = 13.53343    S = 2.353057
CURVE 3 :  L = 22.98044    S = 2.353057
CURVE 4 :  L = 36.19314    S = 2.353057
```

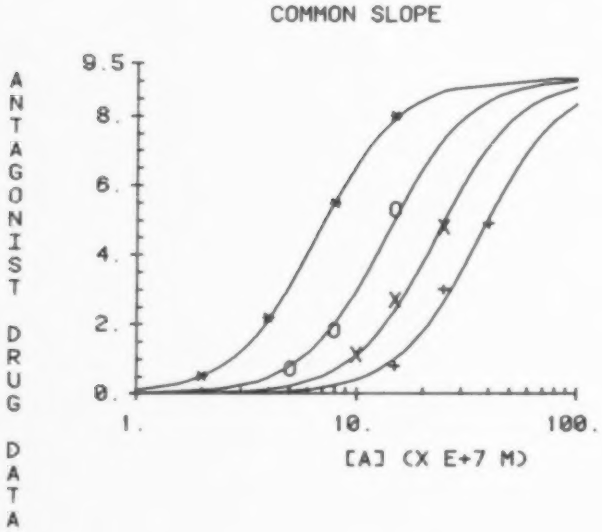


Figure 2. Common slope fit and plot.

Variable Slope Fit

Step 2. To verify that the above procedure is valid, i.e., that the only effect of the antagonist is to shift the K value without affecting S, ANTAGONIST fits the dose-response curves with variable slope-parameters, S_i :

$$E_i = M \cdot \frac{[A]_i^{S_i}}{[A]_i^{S_i} + L_i^{S_i}}$$

ANTAGONIST automatically supplies initial estimates of S_i and L_i in the form of the final converged S and L_i from Step 1. It again displays and prints a graph of the results (Figure 3).

```
LOGISTIC FIT -- VARIABLE SLOPE


CONVERGED VALUES:

CURVE 1 :  L = 6.635501     S = 2.325609
CURVE 2 :  L = 13.33901     S = 2.675337
CURVE 3 :  L = 23.55586     S = 2.135866
CURVE 4 :  L = 36.62731     S = 2.257661
```

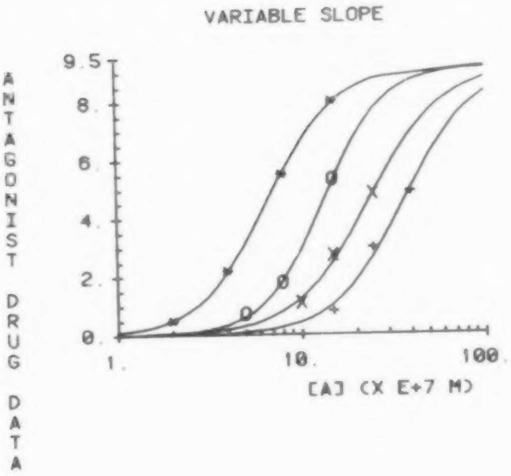


Figure 3. Variable slope fit and plot.

Tests for Non-Parallelism

__Step 3.__ ANTAGONIST makes two tests for non-parallelism of the curves:

- Variance ratio test. The procedure calculates an F statistic from the error sums of squares in Steps 1 and 2. A significant F value indicates that the curves are probably not parallel.

- Range test. Sometimes the F-test indicates that curves are non-parallel when, from a pharmacological standpoint, they may legitimately be considered parallel. For this reason, ANTAGONIST uses an additional test:

$$\%\ \text{slope variation} = 100\left(\frac{\text{largest } S_i - \text{smallest } S_i}{\text{common } S}\right)$$

  In general, a slope variation of up to 10% might be considered acceptable.

ANTAGONIST displays and prints a summary of these tests for non-parallelism, along with the parameters of the fitted functions (Figure 4).

```
          PARAMETERS OF BEST FIT FOR THE FUNCTION

                                M[A]**S
                E = --------------------------
                           [A]**S + L**S

     CURVE DESCRIPTION                          M            L          S
-------------------------------------------------------------------------
A. COMMON SLOPE:

1  *  CONTROL                            9.094E+00    6.594E+00   2.353
2  O  D-TC (1.E-U M)                     9.094E+00    1.353E+01   2.353
3  X  D-TC (2.E-7 M)                     9.094E+00    2.298E+01   2.353
4  +  D-TC (4.E-7 M)                     9.094E+00    3.619E+01   2.353

B. VARIABLE SLOPE:

1  *  CONTROL                            9.144E+00    6.636E+00   2.326
2  O  D-TC (1.E-U M)                     9.144E+00    1.334E+01   2.675
3  X  D-TC (2.E-7 M)                     9.144E+00    2.358E+01   2.136
4  +  D-TC (4.E-7 M)                     9.144E+00    3.663E+01   2.258

             TESTS FOR NON-PARALLELISM

     VARIANCE RATIO TEST :  F(3,4) = .824

     RANGE TEST :  22.9 % VARIATION IN SLOPE
```

Figure 4. Summary of fitted parameters and tests for non-parallelism.

Dose Ratio Regression

<u>Step 4</u>. If the curves are essentially parallel according to the tests in Step 3, ANTAGONIST calculates the dose ratios,

$$DR_i = \frac{L_{i+1}}{L_1}$$

and carries out a linear regression on the equation

$$\log(DR_i - 1) = a + b \log[B]$$

Theoretically, b should be equal to unity and a should be equal to -log K_B. However, the procedure carries out two fits, the first on the above equation and the second with b = 1.

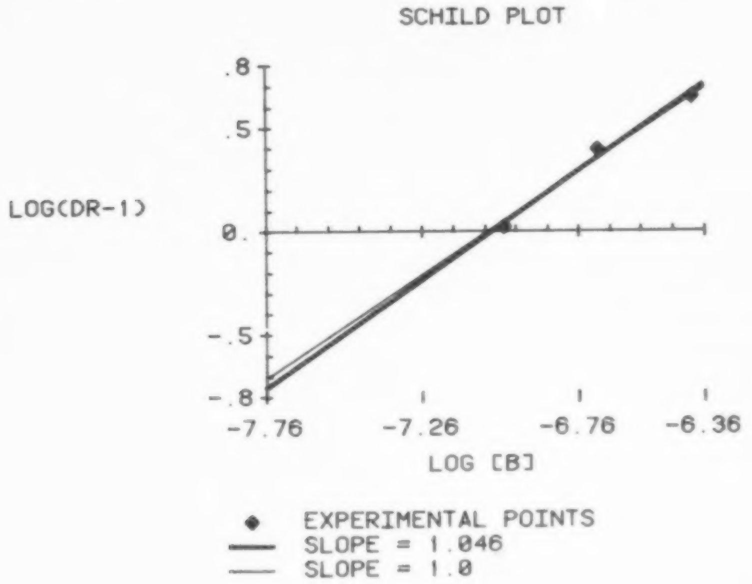
At this point, ANTAGONIST outputs the two estimates of K_B. The procedure does not provide error estimates on K_B since it is more practical to determine K_B for a number of preparations and treat the set of K_B values in the usual way. See Figure 5 for details.

```
DO YOU WISH TO CONTINUE ?YES<


ANTAGONIST CONCENTRATION, CURVE 2 :  1.E-7<
ANTAGONIST CONCENTRATION, CURVE 3 :  2.E-7<
ANTAGONIST CONCENTRATION, CURVE 4 :  4.E-7<

        SUMMARY OF FITS OF THE SCHILD PLOT

LOG(DR-1) VS LOG [B]   (VARIABLE SLOPE)

LOG(DR-1) = 1.046 LOG [B] + 7.366
   CORR. COEFF. = .9944  STD. ERR. = .0475

K(ANTAGONIST) = 9.125E-8
- LOG K = 7.04

LOG(DR-1) VS LOG [B]   (SLOPE = 1)

LOG(DR-1) = LOG [B] + 7.056

K(ANTAGONIST) = 8.799E-8
- LOG K = 7.056
```

Figure 5. Estimates of K_B.

If you used more than one antagonist concentration, the procedure displays and prints a "Schild Plot" of the log (DR - 1) versus log [B] plots (Figure 6). You can create top-level copies of this graph and the two previous ones.

```
DO YOU WISH TO ADD THE CURVE WITH VARIABLE SLOPE
 TO THE DR VS [B] PLOT ?YES<
DO YOU WISH TO ADD THE CURVE WITH SLOPE = 1
 TO THE DR VS [B] PLOT ?YES<
```



```
DO YOU WANT TOP-LEVEL COPIES OF THE ANTAGONIST PLOTS?YES<
    NAME FOR THE COMMON SLOPE PLOT? COMPLOT<
    NAME FOR THE VARIABLE SLOPE PLOT? VARPLOT<
    NAME FOR THE SCHILD PLOT? SCHILD<

RUN TOOK 18.932 CPU SECS>
```

Figure 6. Schild Plot.

Reference

Waud DR and Parker RB. Pharmacological estimation of drug-receptor dissociation constants. Statistical evaluation. II. Competitive antagonists. _J Pharmacol Exp Therap_ 177:13-24, 1971.

LOGISTIC

by Carl Johnson
Mt. Sinai School of Medicine

Summary

LOGISTIC performs a least-squares fit of data to the logistic function

$$Y = \frac{Y_{max} \cdot A^p}{A^p + K^p}$$

where Y_{max}, K, and p are adjustable parameters.

Calling Sequence

```
RAC PUBLIC LOGISTIC <GO>
CALL LOGISTIC(data-table)
              <table>
```

Entry

```
CALL LOGISTIC2(data-table, data-plot)
               <table>     <graph>
```

Citation

Johnson C. LOGISTIC, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Contact Margaret Reid-Miller on the PROPHET line.

Things to Remember

The algorithm used by LOGISTIC is suitable only for curves having the shape shown in Figure 3.

Background and Methods

The logistic function

$$U = \frac{e^{\alpha + \beta r}}{e^{\alpha + \beta r} + 1}$$

can be transformed into a form suitable for the fitting of sigmoidal curves (e.g., log-dose-response curves)

$$Y = \frac{Y_{max} \cdot A^{\nu}}{A^{\nu} + K^{\nu}} \qquad (1)$$

by making the change of variables:

$$\alpha + \beta v = p \cdot \ln(\frac{A}{K})$$

$$U = \frac{Y}{Y_{max}}$$

In this form, the curve reaches a maximum, Y_{max}, has a half-maximal value at K (ED_{50} for a dose-response curve), and has a slope determined by the exponent p.

The public procedure LOGISTIC carries out a least-squares fit of Y versus A data to equation 1 according to the procedure of Waud and Parker (1971).

Input and Interaction

You can simultaneously fit as many curves as you like. However, LOGISTIC forces all of the curves to have the same Y_{max} value. Thus, if you have data for two full agonists, it is reasonable to fit them simultaneously; but if you have data for a full agonist and a partial agonist, you should fit the two curves separately by calling the procedure twice. To use LOGISTIC, type

```
CALL LOGISTIC(data-table)
```

where data-table is the name of the table containing the data. The procedure asks how many curves you want to fit, which columns contain the X values (A in equation 1), and which columns contain the Y values (Y in equation 1).

LOGITDATA: 2*CELL (COL 1) 7R X 4C

	1 [NLE#3] IN UL	2 EVENT #2	3 EVENT #4-#3	4 EVENT #6-#5
1	5.	.1	.1	0.0
2	10.	1.0	.8	0.0
3	20.	1.7	2.0	.8
4	40.	2.0	2.4	1.5
5	80.	2.5	3.0	2.1
6	160.		3.1	2.7
7	320.			3.0

Figure 1. A typical LOGISTIC data table.

LOGISTIC then asks whether you want to examine a semilog plot of the data and whether you want automatic convergence. For automatic convergence, you must supply a convergence value expressed as a percentage. For example, if you want the iterations carried out until all values of K and P are less than 0.1% different from the values computed on the previous iteration, you enter a convergence value of 0.1. (For a more complete description of how this automatic convergence works, see its description in the public procedure AGONIST.)

Whether or not you choose automatic convergence, you must provide initial estimates of the adjustable parameters Y_{max}, P, and K for each curve.

Iterative Least-Squares Fit

LOGISTIC then carries out an iterative least-squares fit. If you chose the automatic convergence option, the iterations continue until the current value differs from the value of the previous iteration by less than the convergence value.

If you did not choose the automatic convergence option, you must answer LOGISTIC's question,

CONTINUE?

at each iteration. If you answer YES, iteration continues. If you answer NO, iteration stops and the procedure continues. If you answer RESTART, LOGISTIC prompts you for new initial estimates.

```
>RAC PUBLIC LOGISTIC<
>CALL LOGISTIC(LOGITDATA)<


NUMBER OF CURVES TO BE FIT = 3<

COLUMN CONTAINING X-VALUES = 1<
 COLUMN NUMBER FOR CURVE 1 = 2<
 COLUMN NUMBER FOR CURVE 2 = 3<
 COLUMN NUMBER FOR CURVE 3 = 4<

DO YOU WANT TO EXAMINE A GRAPH OF THE DATA?NO<

DO YOU WISH TO USE AUTOMATIC CONVERGENCE?YES<
    CONVERGENCE FACTOR (AS A  PERCENT) = .1<
```

Figure 2. Interacting with LOGISTIC.

The iterative least-squares fit occurs in two stages. In the first stage, all curves are forced to have the same slope parameter, p (i.e., all curves are parallel). At the end of this stage, LOGISTIC displays a graph of the curves with the common slopes (Figure 3).

```
                  LOGISTIC FIT -- COMMON SLOPE

INITIAL ESTIMATE OF SLOPE = 1.<

INITIAL ESTIMATE OF K, CURVE 1 = 10.<
INITIAL ESTIMATE OF K, CURVE 2 = 12.<
INITIAL ESTIMATE OF K, CURVE 3 = 22.<


                       CONVERGED VALUES:


CURVE 1 :  K = 19.77701     SLOPE = 1.652308
CURVE 2 :  K = 16.23894     SLOPE = 1.652308
CURVE 3 :  K = 43.1157      SLOPE = 1.652308
```

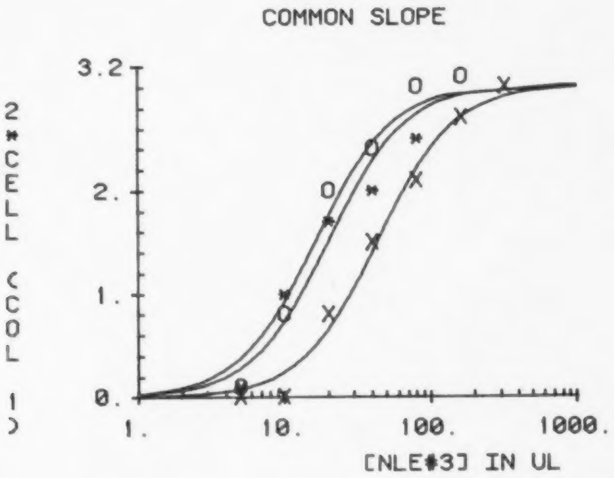


Figure 3. Common slope fit and plot.

In the second stage, each curve is allowed to have its own best slope.

LOGISTIC prints a plot of the variable slopes (Figure 4). It also prints a summary of the fitted parameters and the tests for parallelism (Figure 5).

```
                LOGISTIC FIT -- VARIABLE SLOPE


                      CONVERGED VALUES:


CURVE 1 :  K = 20.05868      SLOPE = 1.307183
CURVE 2 :  K = 15.96735      SLOPE = 2.15403
CURVE 3 :  K = 42.67491      SLOPE = 1.675679
```

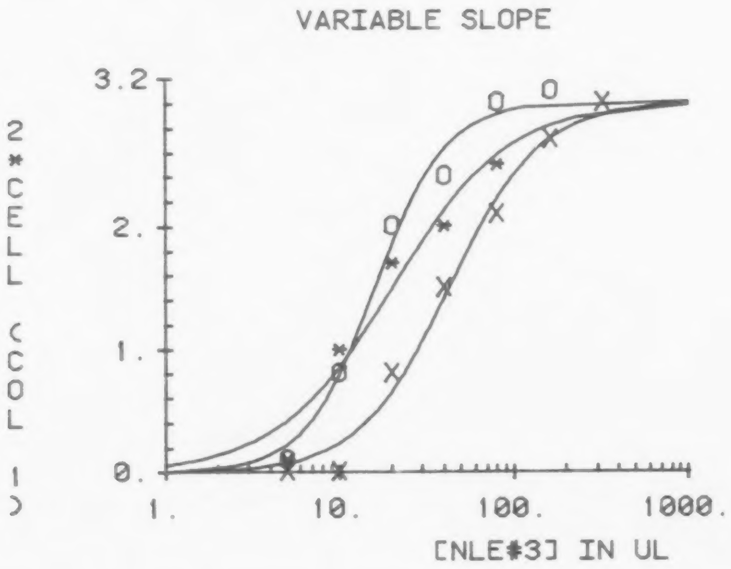


Figure 4. Variable fit and plot.

```
              PARAMETERS OF BEST FIT FOR THE FUNCTION
                                  EMAX
                    E =  --------------------------
                            1  +  (K/[A])**N

        CURVE DESCRIPTION                          EMAX          K          N
-------------------------------------------------------------------------------
A. COMMON SLOPE:

1  *  EVENT #2                                  .302E+01    .198E+02   1.652
2  O  EVENT #4-#3                               .302E+01    .162E+02   1.652
3  X  EVENT #6-#5                               .302E+01    .431E+02   1.652

B. VARIABLE SLOPE:

1  *  EVENT #2                                  .301E+01    .201E+02   1.307
2  O  EVENT #4-#3                               .301E+01    .160E+02   2.154
3  X  EVENT #6-#5                               .301E+01    .427E+02   1.676


                  TESTS FOR NON-PARALLELISM

     VARIANCE RATIO TEST :  F(2,11) = 2.500294

     RANGE TEST :  51.25238 % VARIATION IN SLOPE
```

Figure 5. Summary of fitted parameters and tests for parallelism.

Quick Entry LOGISTIC2

If you already have a graph of the curves on which you want to do a logistic fit, you can save time by using the entry LOGISTIC2. Type

```
CALL LOGISTIC2(data-table, data-plot)
```

where data-plot is the name of your graph.

Reference

Waud DR and Parker RB. Pharmacological estimation of drug-receptor dissociation constants. Statistical evaluation. J Pharmacol Exp Ther 177:1-24, 1971.

ABSORB

by Theodor W. Guentert
University of California Medical Center
School of Pharmacy at San Francisco

Summary

ABSORB calculates the values needed for a representation of a drug absorption profile (Wagner-Nelson or Loo-Riegelman method). It interactively plots the absorption profile and tests for zero-order or first-order input.

Calling Sequence

```
RAC PUBLIC ABSORB <GO>
CALL ABSORB <GO>
```

Entry

```
CALL ABSORBQT(data-table,  parameters)
                <table>      <float>
                               or
                           <float array>
```

Citation

Guentert TW. ABSORB, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things to Remember

ABSORB changes the input data table so that it contains the information needed for plotting absorption profiles. You can use ABSORB to prepare this data and/or to prepare linear-linear and log-linear plots with selected windows, allowing linear regression over a chosen range.

Given a set of concentration-time data, the public procedure ABSORB computes the values needed for plotting a drug absorption profile. Depending on whether the drug follows one- or two-compartment kinetics, ABSORB uses either the Wagner-Nelson or the Loo-Riegelman approach. Once it has computed the absorption values, the procedure enables you to prepare various plots of the data.

ABSORB's Computations

At each time point, ABSORB computes

- the area under the concentration-time curve up to that time (AUC_0^T)
- the concentration in the tissue compartment for drugs following two-compartment kinetics (CT)
- the amount absorbed up to that time divided by the volume of the central compartment (A/V1)
- the midpoint of each sampling interval (TMID)
- linear and logarithmic increments (D(A/V1)/DT)
- decrements in the amount remaining to be absorbed, expressed as a percentage (%ARAB)

ABSORB can then use these values to make linear-linear and log-linear plots of %ARAB versus time and/or D(A/V1)/DT versus TMID.

Interacting with ABSORB

When you call ABSORB, the procedure asks you for the following information:

- the name of the table containing time values in column 1 and concentration values in column 2
- whether the drug follows one- or two-compartment kinetics
- for drugs with one-compartment kinetics (Wagner-Nelson approach), the elimination rate constant

- for drugs with two-compartment kinetics (Loo-Riegelman approach), the following parameters:

ALPHA (fast disposition rate constant)
BETA (slow disposition rate constant)
K_{21} (rate constant for transfer of drug from second to first compartment)

ABSORB then computes the information it needs to plot a drug absorption profile, adds the values to the data table you named, displays the completed table, and asks whether you would like to prepare some plots.

You can prepare two different kinds of plots, each of which will be displayed linear-linear and log-linear:

- %ARAB versus time
- Increments of D(A/V1)/DT versus TMID

In both cases, ABSORB asks whether you want to look at a "window" of the plots. (Consult the example at the end of this description for illustration of the plots produced by ABSORB.) The procedure also prompts you for the points to be used in calculating unweighted least-squares linear regression lines. If the procedure achieves an acceptable fit to the chosen points in the linear-linear or log-linear plot, the slope of the regression line (given as the curve label) is indicative of the zero-order or the first-order input rate, respectively.

Quiet Entry
ABSORBQT

ABSORB also has a quiet entry, ABSORBQT. Use ABSORBQT to call the procedure in a batch job or within another procedure to compute the information needed to make plots of the absorption profile. The command is:

```
CALL ABSORBQT(data-table, parameters)
```

Data-table is the table to which you want to apply a Wagner-Nelson or Loo-Riegelman treatment.

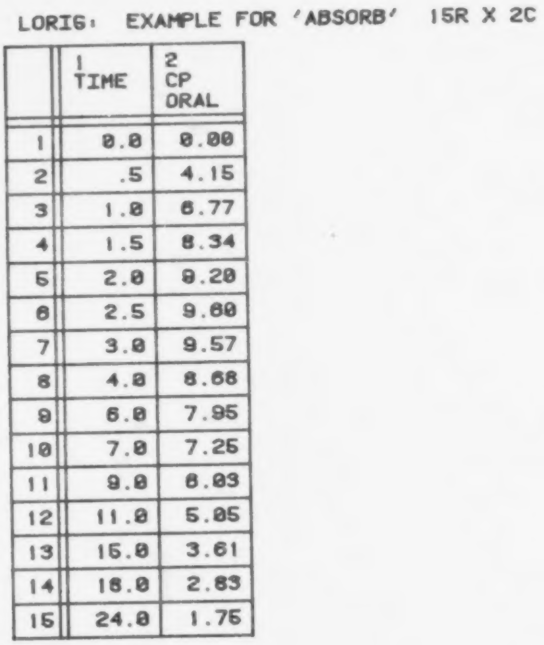
The second argument, parameters, can be either a FLOAT or a FLOAT ARRAY. If the drug obeys one-compartment kinetics, you should enter the elimination rate constant as a FLOAT variable. If a two-compartment model applies, enter ALPHA, BETA, and K_{21} ,in that order, as a FLOAT ARRAY.

Once the data-table is completed, you can call ABSORB and use it to make plots without going through the initial interactive quiz.

References

Loo JCK, Riegelman S. A New Method for Calculating the Intrinsic Absorption Rate of Drugs. _J Pharm Sci_ 57:918-28, 1968.

Wagner JG, Nelson E. Percent Absorbed Time Plots Derived from Blood Level or Urinary Excretion Data. _J Pharm Sci_ 52:610-11, 1963.



LORIG: EXAMPLE FOR 'ABSORB' 15R X 2C

	1 TIME	2 CP ORAL
1	0.0	0.00
2	.5	4.15
3	1.0	6.77
4	1.5	8.34
5	2.0	9.20
6	2.5	9.60
7	3.0	9.57
8	4.0	8.68
9	6.0	7.95
10	7.0	7.25
11	9.0	6.03
12	11.0	5.05
13	15.0	3.61
14	18.0	2.83
15	24.0	1.75

Figure 1. A typical data table supplied as input to ABSORB.

```
>RAC PUBLIC ABSORB<
>CALL ABSORB<

WHAT IS THE NAME OF YOUR TABLE WITH TIME-VALUES IN
COL 1 AND CP-VALUES IN COL 2?LORIG<
DOES THE DRUG FOLLOW 1-COMPARTMENT-KINETICS?NO<
PLEASE ENTER ALPHA-VALUE?.68<
PLEASE ENTER BETA-VALUE?.08<
PLEASE ENTER K21-VALUE?.31<
```

Figure 2. Calling the procedure.

LORIG: EXAMPLE FOR 'ABSORB' 15R X 10C

	1 TIME	2 CP ORAL	3 AUC 0->T	4 CT	5 XARAB	6 (D) LN(XARAB)	7 A/V1	8 TMID	9 D(A/V1)/DT	10 (D) LN D(A/V1)/DT
1	0.0	0.00	0.	0.	100.	4.60517	0.			
2	.5	4.15	1.0375	.2848105	81.36411	4.398934	4.616875	.25	9.23375	2.222866
3	1.0	6.77	3.7675	.9289096	66.25491	4.19351	8.360045	.75	7.486341	2.01308
4	1.5	8.34	7.545	1.755257	53.90636	3.987249	11.41928	1.25	6.118475	1.811313
5	2.0	9.20	11.93	2.618848	43.8431	3.780617	13.91237	1.75	4.986175	1.606669
6	2.5	9.60	16.63	3.438503	35.59082	3.572088	15.9568	2.25	4.088858	1.408266
7	3.0	9.57	21.4225	4.163464	29.39093	3.380666	17.49277	2.75	3.071935	1.122308
8	4.0	8.68	30.54026	5.200442	22.33922	3.106344	19.23977	3.5	1.746998	.557899
9	6.0	7.95	47.15967	6.182266	9.550691	2.256614	22.40801	5.	1.584122	.4600304
10	7.0	7.25	54.75419	6.322757	6.429587	1.86091	23.18124	6.5	.7732258	-.2571842
11	9.0	6.03	67.99675	6.084935	2.933873	1.076324	24.04727	8.	.4330159	-.8369808
12	11.0	5.05	79.04779	5.51360	1.36762	.3130721	24.43529	10.	.1940128	-1.639832
13	15.0	3.61	96.20697	4.204506	.310117	-1.170806	24.69728	13.	.06549674	-2.725755
14	18.0	2.83	105.8195	3.345013	.1189614	-2.128956	24.74464	16.5	.01578669	-4.148651
15	24.0	1.75	119.301	2.084364	.017544	-4.043043	24.76976	21.	.004187544	-5.475841

Figure 3. Data-table after ABSORB has added computed absorption values.

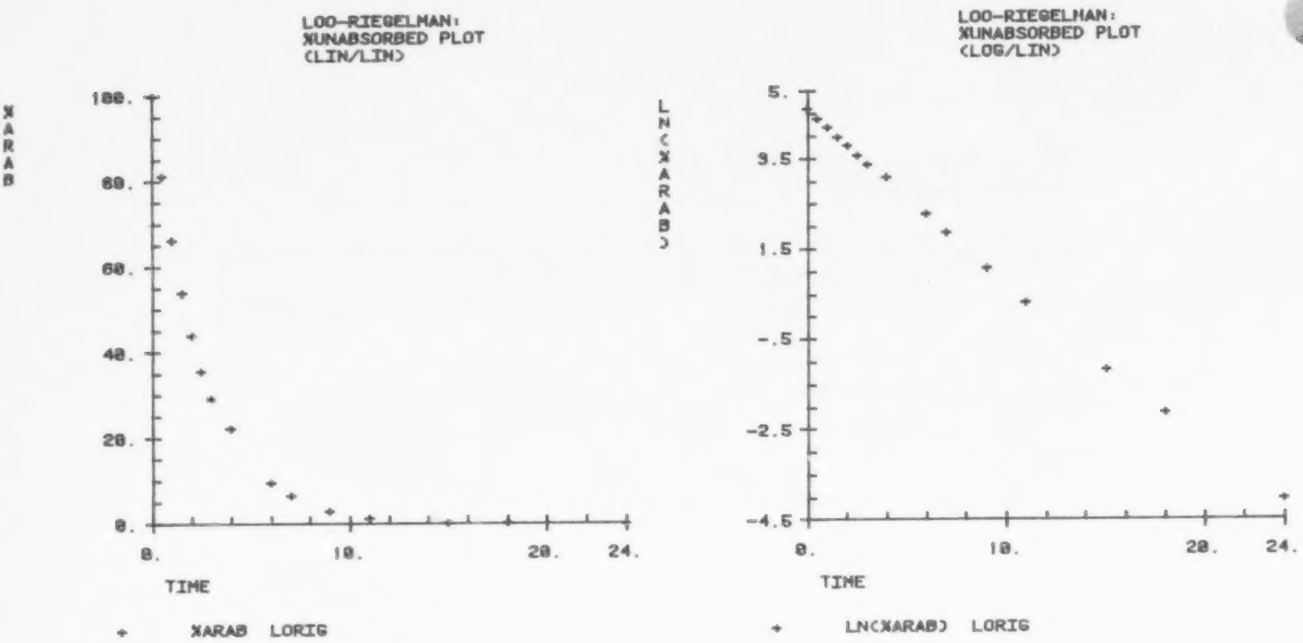


Figure 4. Linear-Linear and Log-Linear plots of %ARAB versus time.

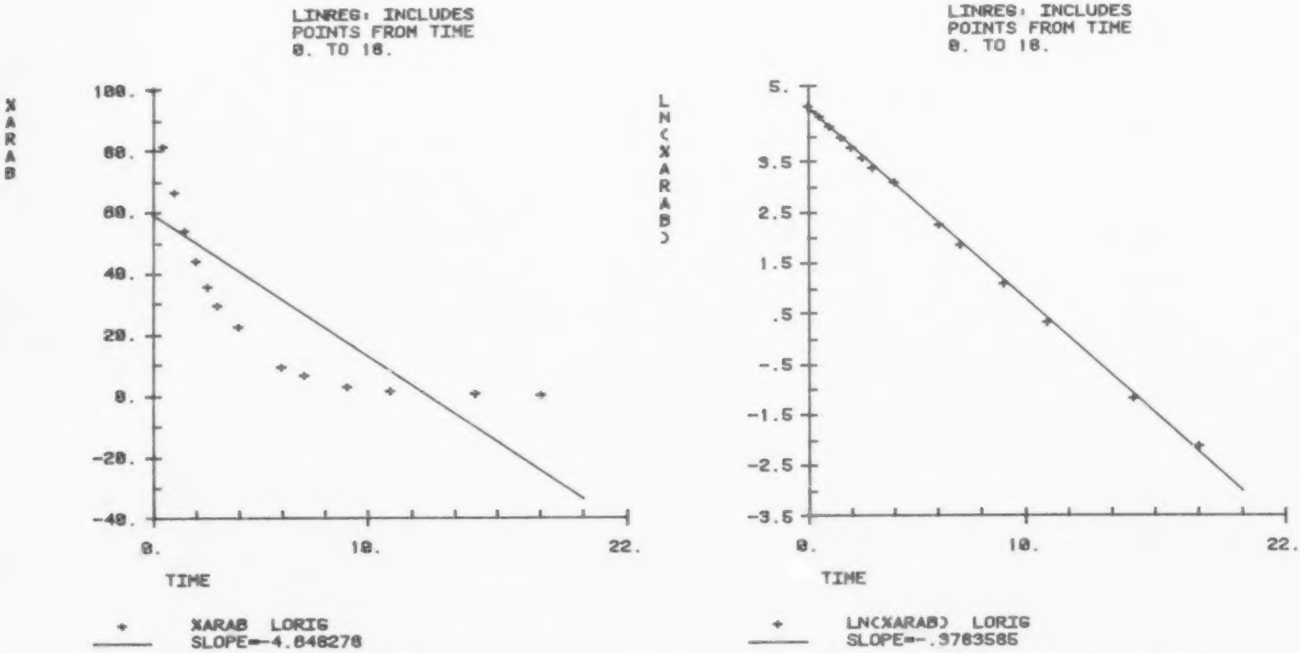


Figure 5. Interaction with ABSORB and the ensuing windows of plots of fitted lines to %ARAB versus time.

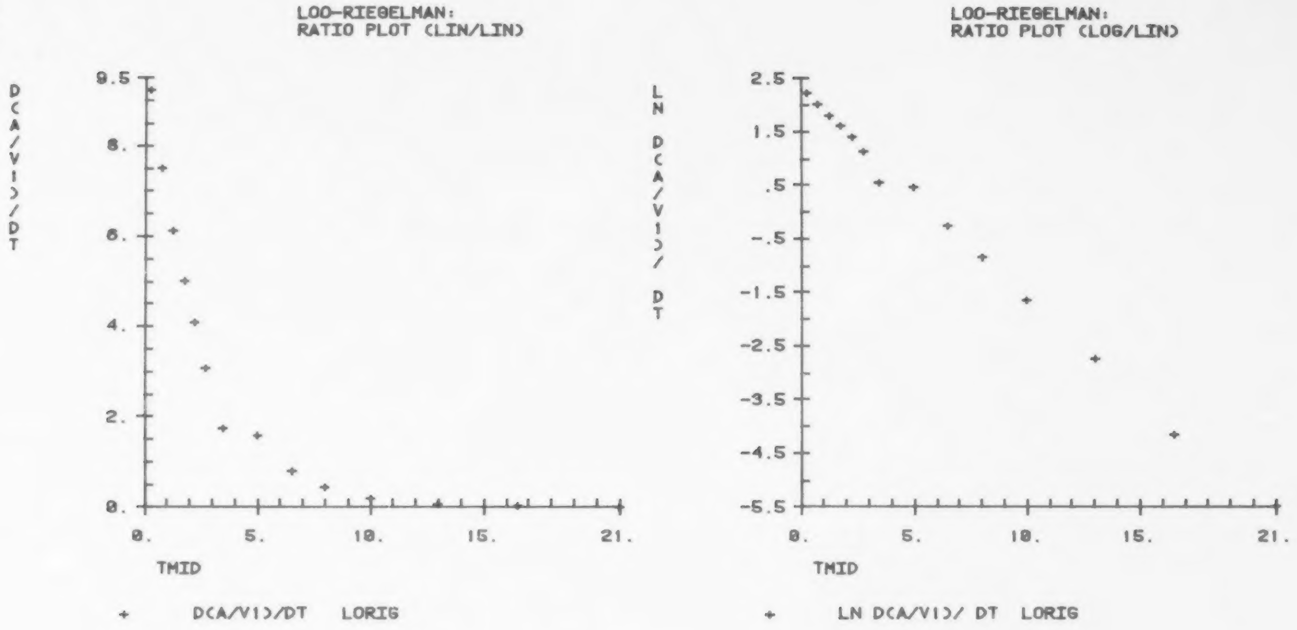


Figure 6. Linear-Linear and Log-Linear plots of increments of D(A/V1)/DT versus TMID.

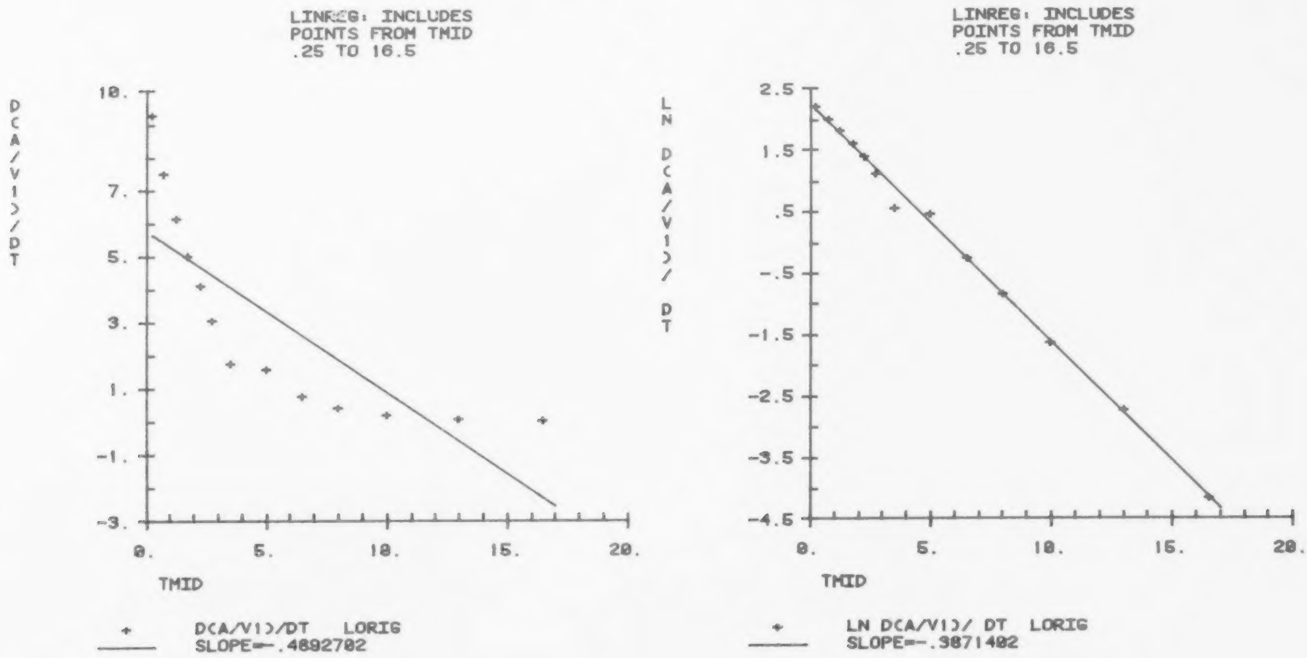


Figure 7. Interaction with ABSORB and the ensuing windows of plots of fitted lines to increments of D(A/V1)/DT versus TMID.

DIFFEQ

by Harold Perry
Bolt Beranek and Newman Inc.

Summary

DIFFEQ uses MLAB to fit the parameters of a system of differential equations and/or to integrate numerically a system of differential equations. The system must be in terms of only one independent variable.

Calling Sequence

```
RAC PUBLIC DIFFEQ <GO>
CALL FITDIFF <GO>

RAC PUBLIC DIFFEQ <GO>
CALL INTDIFF <GO>

RAC PUBLIC DIFFEQ <GO>
CALL FITANDINTDIFF <GO>
```

Entries

```
     CALL FITDIFFQT(function-table, parameter-table, data-table,
                      <table>          <table>         <table>

                  ind-var-col,     par-vary,       par-consts,
                    <fixed>      <text array>     <text array>

                       fit-ans,    tab-opts,   tab-results)
                    <float array>  <fixed>    <table array>

     CALL INTDIFFQT(function-table, parameter-table, ind-var-name,
                      <table>           <table>          <text>

                       ranges,          int-results)
                  <float array array>    <table>

     CALL FITANDINTDIFFQT(function-table, parameter-table, data-table,
                              <table>          <table>         <table>

                          ind-var-col,     par-vary,       par-consts,
                            <fixed>      <text array>     <text array>

                               fit-ans,    tab-opts,   tab-results,
                            <float array>  <fixed>    <table array>

                               ranges,          int-results)
                          <float array array>    <table>
```

Citations

MLAB, An On-Line Modelling Laboratory, 7th ed. Division of Computer Research and Technology, NIH, Bethesda, MD, 1977.

Perry H. DIFFEQ, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things to Remember

Convergence of the MLAB curve-fitting algorithm does not necessarily mean that you have a good fit. It only means that MLAB has done the best it can with the given specifications of the fit. You must use the statistical measures (e.g., F-values, sum of squares, a graph of the fit) to determine how good the fit really is.

In contrast to FITFUN (described in PROPHET Statistics), DIFFEQ requires a data table that does not contain any EMPTY entries.

DIFFEQ does not check whether any constraints or differential equations are syntactically correct. (See the MLAB documentation for further details.)

In the mathematical modelling of biological systems, a number of models can be described by a set of differential equations. Compartmental modeling is a typical example. In such a model, the concentrations of a drug in each of several compartments in the body are described by separate differential equations that may involve initial concentration, concentration in other compartments, rate constants, and other parameters.

Given the differential equations and boundary conditions that define a model, you may want to let the model run for some period of time with a given set of parameter values and track the values of various functions. On the other hand, if you have experimental data (perhaps weighted) on the variation of one or more of the functions over time, you may want to fit the values of the parameters to match the behavior of the function over time with the experimental values. The first process is equivalent to the numerical integration of the sytem of differential equations; the second process involves fitting the parameters in the system to the experimental data.

Input

The public procedure DIFFEQ calls MLAB, which has facilities for fitting parameters and integrating systems of differential equations. MLAB applies a Marquardt-Levenberg iterative curve-fitting algorithm. The parameterized equations may include any combination of powers, logarithms, trigonometric functions, and hyperbolic functions. Parameter names and the independent variable name may be any legal PROPHET variable names.

Convergence of the Fitting Algorithm

MLAB decides whether or not the fitting algorithm has converged based on the improvement in the sum of squares error from one iteration to the next. If the ratio of the current sum of squares error to the previous one is greater than the convergence factor, then the fit is considered still improving and therefore not converged. The fit is considered converged only when the algorithm cannot improve the parameter values any further and thus has done the best it can with the given data.

CONVERGENCE OF THE ALGORITHM THEREFORE HAS NO RELATION TO THE GOODNESS OF FIT OF THE FITTED FUNCTIONS, WHICH SHOULD BE DETERMINED BY THE STANDARD DEVIATIONS AND OTHER MEASURES.

Parameter Constraints

You can enforce constraints on the parameters to fit. Consult the documentation of FITFUN for the syntax of legal constraints.

IF ANY OF THE FITTED PARAMETERS IS AT THE BOUNDARY OF A CONSTRAINT, THEN MOST OF THE STATISTICS PRESENTED AFTER THE FIT (e.g., F-VALUES, MULTIPLE R^2) ARE NOT MEANINGFUL.

You should use constraints only when you know some of the properties of the parameters you wish to constrain. The resultant fits should provide good initial values for the unconstrained parameters that you know little about. After performing a series of constrained fits to get good initial values for all the parameters, you should fit the system with no constraints. The results of this fit will be the results you use to measure the goodness of the overall fit.

Output

The procedure's output is almost identical to that of FITFUN, which includes a parameter table with the fitted parameter values, their standard deviations, and their significance levels. DIFFEQ will also present the standard deviation of regression, the multiple correlation coefficient R^2, plus an analysis-of-variance table. If you enforced constraints, a table of LaGrange multipliers will be displayed. You can also see a table or graphs of the residuals plus a graph of the fit for each differential equation, and a table of integrated function values over varying ranges.

Interactive Entries

MLAB provides the best fits to your system when the initial parameter values are well specified. Assuming that your system is a good model of the observed data, your objective should be to modify the parameter specifications in the hope of getting as good a fit as desired. DIFFEQ is designed to allow you to accomplish this goal in a convenient, straightforward manner, and you can save and compare the fitted parameter values of various fits.

Once you are satisfied with the fitted values of the parameters of your system, you can use DIFFEQ to integrate the system.

A summary of each mode of interaction follows. Consult the examples for further elaboration.

FITDIFF

This entry lets you fit the parameters of your system. To use FITDIFF, type the command:

```
CALL FITDIFF <GO>
```

FITDIFF requires a function table and a parameter table to represent conveniently the system you wish to fit. The parameter table also serves as a database for comparing the fitted parameter values of various fits.

Once you invoke FITDIFF, you can edit your function and parameter tables if they are in your directory. If they are not, the procedure allows you to construct them. The form of each table is described below.

Function Table

Information about each differential equation in your system is saved as a row in the function table:

- Column 0 contains the TEXT name of the function.
- Column 1 contains the TEXT differential equation of the function.
- Column 2 contains the TEXT boundary value of the function.
- Column 3 contains the TEXT value of the independent variable at which the function assumes its boundary value.
- Note that columns 2 and 3 may contain the names of parameters.

Parameter Table

The parameter table contains the name of each parameter in the system plus a corresponding initial value that is used to initiate a fit. Column 0 contains the parameter names; column 1, their initial values. Sample function and parameter tables are shown in Figure 1. Row 1 of FNTAB is interpreted as:

$$dY1/dT = -K1 * Y1 \quad \text{such that } Y1(0) = A$$

Figure 1 also contains a schematic of the system represented by FNTAB at time T = 0.

best copy available

FNTAB 3R X 3C

0 FUNCTION NAME	1 DERIVATIVE	2 BOUNDARY VALUE OF FUNCTION	3 AT T =
1. Y1	-K1*Y1	A	0
2. Y2	K1*Y1-K2*Y2	0	0
3. Y3	K2*Y2-K3*Y3	0	0

PARAMTAB 4R X 1C

0 PARAMETER NAME	1 INITIAL VALUE
1. K1	1.
2. K2	1.
3. K3	1.
4. A	2.

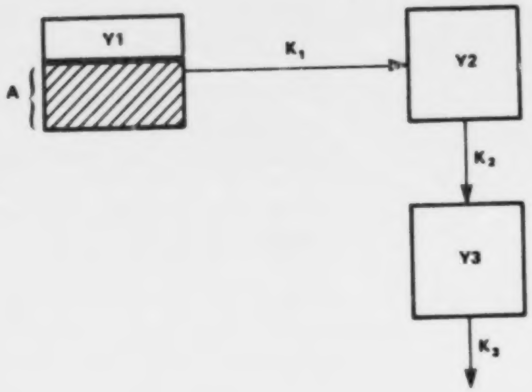


Figure 1. A function table (upper left) and a parameter table (upper right) used to represent a 3-compartment model.

Other Input

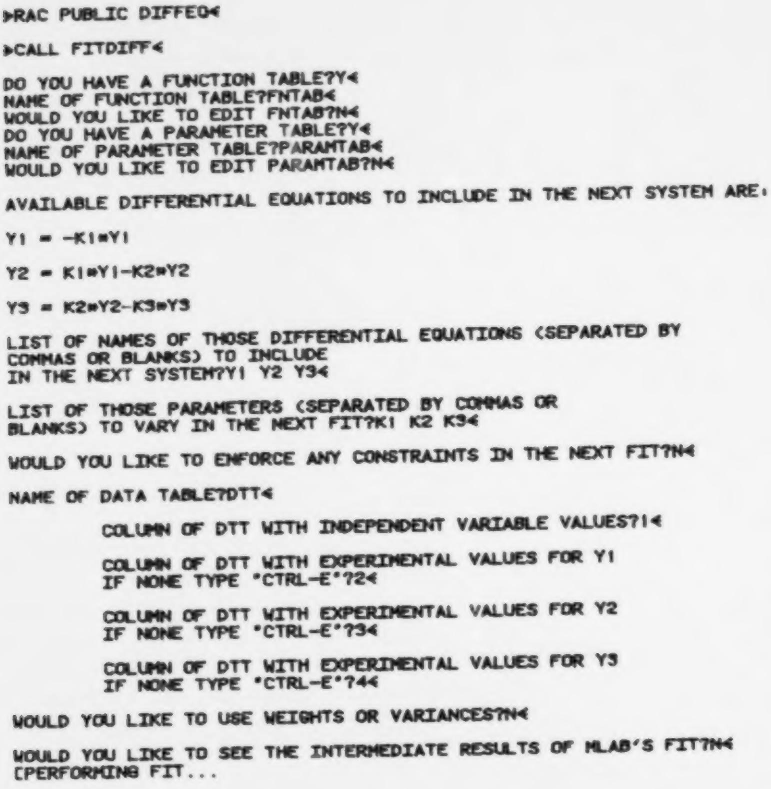
To perform a fit, FITDIFF needs the following information:

- the names of the differential equations to include in the next system to fit
- which parameters to vary in the next fit. (If a parameter does not vary, the procedure sets it equal to its initial value throughout the fit.)
- any constraints desired
- the column number of your parameter table containing initial parameter values
- the name of your data table
- the column number of your data table containing independent variable values

- the column number of your data table containing experimental function values
- the column number of your data table containing weights or variances (if you wish to use them)

FITDIFF prompts you for this information in a "friendly" manner; it asks unambiguous questions and provides information to help you answer them.

Before invoking MLAB to perform a fit, FITDIFF also asks whether you would like to see MLAB's intermediate fit results. (Consult the <u>MLAB Manual</u> cited in the summary box for illustration of these results.) Figure 2 illustrates your preliminary interaction with FITDIFF.



```
▶RAC PUBLIC DIFFEQ◀

▶CALL FITDIFF◀

DO YOU HAVE A FUNCTION TABLE?Y◀
NAME OF FUNCTION TABLE?FNTAB◀
WOULD YOU LIKE TO EDIT FNTAB?N◀
DO YOU HAVE A PARAMETER TABLE?Y◀
NAME OF PARAMETER TABLE?PARAMTAB◀
WOULD YOU LIKE TO EDIT PARAMTAB?N◀

AVAILABLE DIFFERENTIAL EQUATIONS TO INCLUDE IN THE NEXT SYSTEM ARE:

Y1 = -K1*Y1

Y2 = K1*Y1-K2*Y2

Y3 = K2*Y2-K3*Y3

LIST OF NAMES OF THOSE DIFFERENTIAL EQUATIONS (SEPARATED BY
COMMAS OR BLANKS) TO INCLUDE
IN THE NEXT SYSTEM?Y1 Y2 Y3◀

LIST OF THOSE PARAMETERS (SEPARATED BY COMMAS OR
BLANKS) TO VARY IN THE NEXT FIT?K1 K2 K3◀

WOULD YOU LIKE TO ENFORCE ANY CONSTRAINTS IN THE NEXT FIT?N◀

NAME OF DATA TABLE?DTT◀

        COLUMN OF DTT WITH INDEPENDENT VARIABLE VALUES?1◀

        COLUMN OF DTT WITH EXPERIMENTAL VALUES FOR Y1
        IF NONE TYPE 'CTRL-E'?2◀

        COLUMN OF DTT WITH EXPERIMENTAL VALUES FOR Y2
        IF NONE TYPE 'CTRL-E'?3◀

        COLUMN OF DTT WITH EXPERIMENTAL VALUES FOR Y3
        IF NONE TYPE 'CTRL-E'?4◀

WOULD YOU LIKE TO USE WEIGHTS OR VARIANCES?N◀

WOULD YOU LIKE TO SEE THE INTERMEDIATE RESULTS OF MLAB'S FIT?N◀
[PERFORMING FIT...
```

Figure 2. Supplying FITDIFF with the information it needs to perform a fit.

Output

FITDIFF now invokes MLAB to perform the specified fit. If you see an asterisk (*) before any table results, then for some reason, MLAB could not understand your intentions and aborted. If this happens, type:

```
EXIT <GO>
```

This returns you to PROPHET, where you can examine the list of commands sent to MLAB. If no asterisk appears, FITDIFF presents the following results:

- a parameter results table, which includes for each varied parameter, its fitted value, standard deviation, t-value, and significance level. (See Figure 3.)
- a summary of the fit, which includes an analysis-of-variance data table and other statistics (See Figure 3.)

For each differential equation in the fitted system, FITDIFF presents:

- an optional graph of the fit (See Figures 4, 6, and 8.)
- an optional table of residuals, which you can acquire as a TEMPORARY table (See Figure 5.)
- three optional graphs of the residuals (See Figures 7 and 9.):
    1) residuals versus the independent variable
    2) residuals versus the fitted dependent variable
    3) a histogram of the residuals' distribution

Consult Figures 3 through 9 for details of FITDIFF's post-fit interaction and output.

```
FIT FOR D.E. Y1 = -K1*Y1 TO:
COLUMN 1 VS COLUMN 2 OF DTT
FIT FOR D.E. Y2 = K1*Y1-K2*Y2 TO:
COLUMN 1 VS COLUMN 3 OF DTT
FIT FOR D.E. Y3 = K2*Y2-K3*Y3 TO:
COLUMN 1 VS COLUMN 4 OF DTT

NUMBER OF DATA POINTS = 48
MULTIPLE R-SQUARED = .9997499
STANDARD DEVIATION OF REGRESSION = .006255258
FITLAB USED 4 ITERATION(S) AND CONVERGED
```

PARAMETER TABLE

PARAMETER	FITTED VALUE	STANDARD DEVIATION	T-VALUE	SIG. LEV.
K1	1.003954	.002528505	397.0542	.0001
K2	1.001125	.003489576	286.8902	.0001
K3	.5016227	.00318925	157.2855	.0001

ANALYSIS OF VARIANCE TABLE (NOTES)

SOURCE	SUM OF SQUARES	D.F.	MEAN SQUARE	F VALUE	SIG. LEV.
REGRESSION	7.038265	2.	3.519132	89938.39	.0001
RESIDUAL	.001760772	45.	3.912826×10^{-5}		

```
RESIDUAL SUM OF SQUARES FOR: Y1 = .00144347
                             Y2 = .0001638657
                             Y3 = .0001534363
```

Figure 3. Parameter results table and a table of the analysis of variance summary.

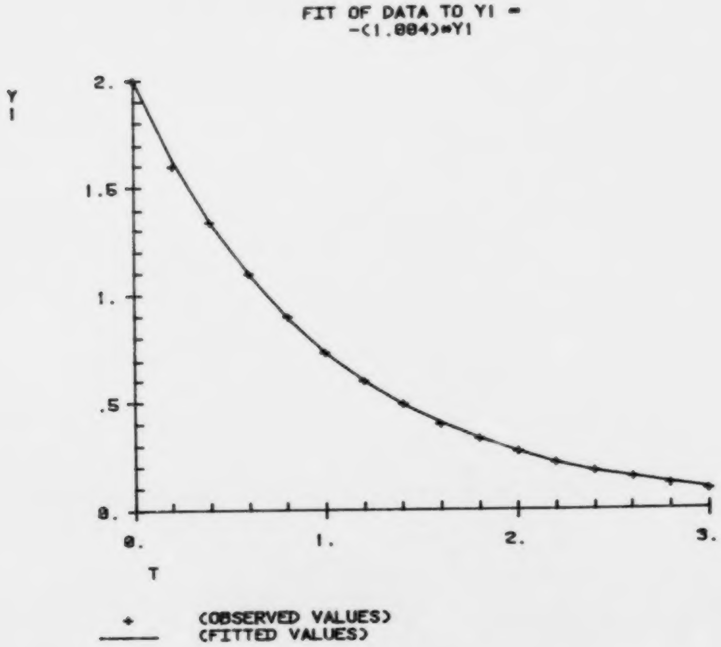


Figure 4. Graph of Y1's fit.

TABLE OF RESIDUALS FOR Y1

T	Y1 (OBS)	Y1 (FITTED)	RESIDUAL
0.	2.	2.	0.
.2	1.6	1.636161	-.03616104
.4	1.34	1.338517	.001482669
.6	1.1	1.095922	.004077554
.8	.9	.8958221	.004177908
1.	.735	.7328572	.002142824
1.2	.6	.5995396	.0004603565
1.4	.49	.4904749	-.0004749149
1.6	.4	.4012498	-.001249806
1.8	.33	.3282561	.001743909
2.	.27	.2685417	.001458317
2.2	.22	.2196898	.0003101589
2.4	.18	.1797246	.0002753906
2.6	.15	.1470902	.002909794
2.8	.12	.1202833	-.0002832515
3.	.09	.09640168	-.006401676

Figure 5. Y1's table of residuals.

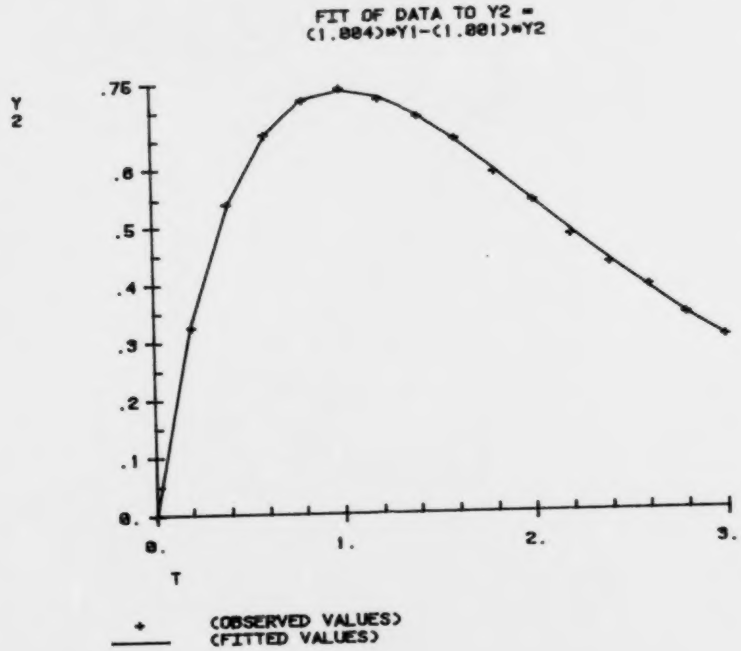


Figure 6. Graph of Y2's fit.

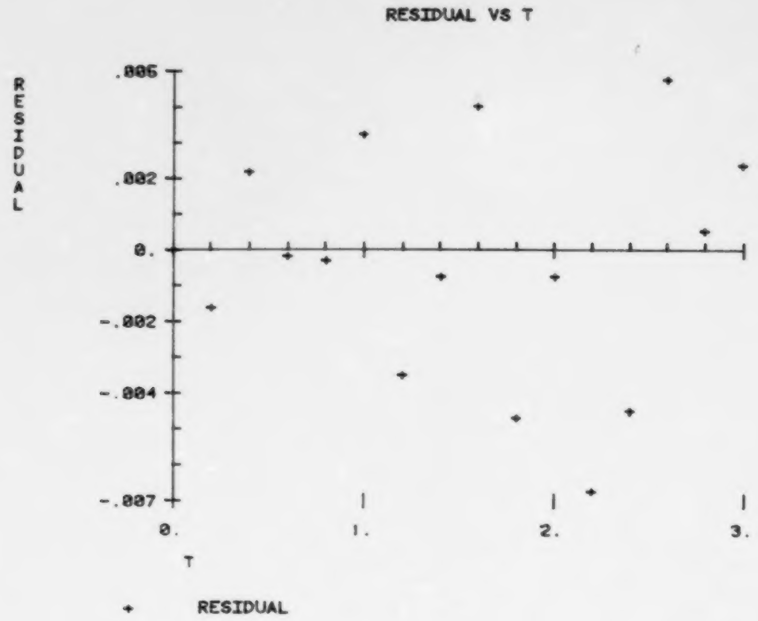


Figure 7. Graph of Y2's residuals versus T, the independent variable.

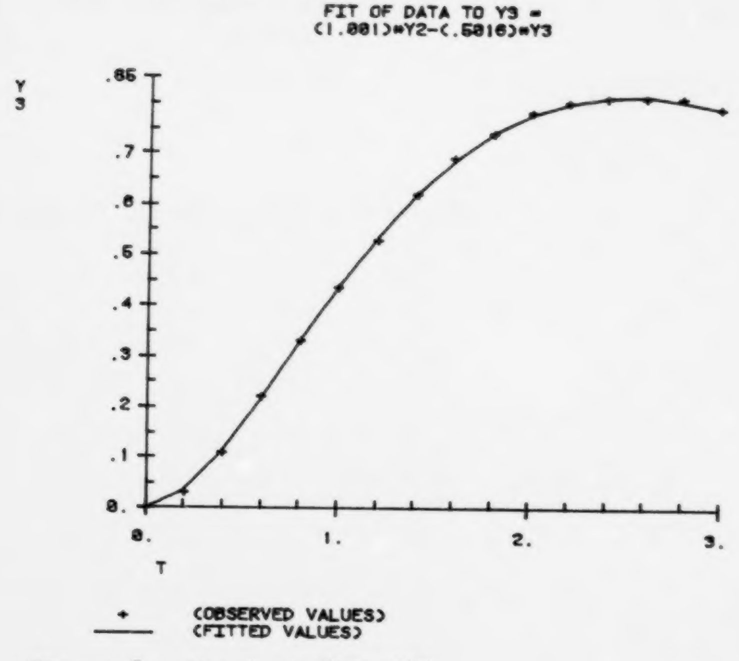


Figure 8. Graph of Y3's fit.

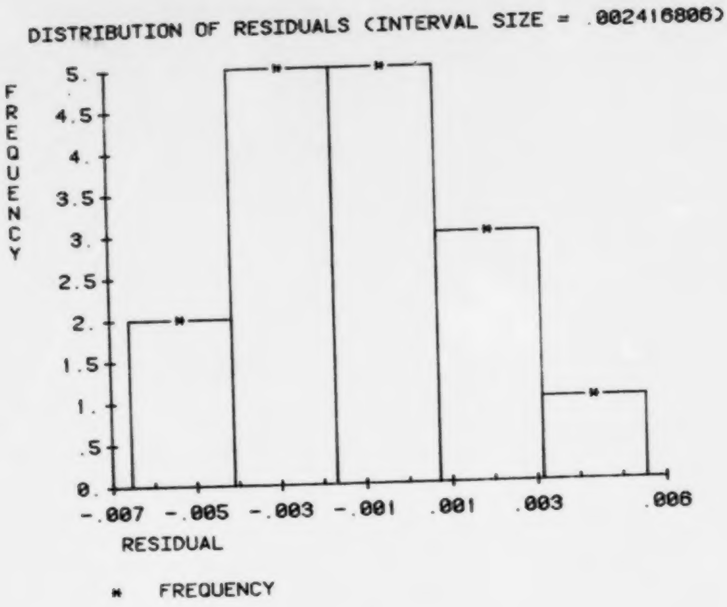


Figure 9. Histogram of the residuals of Y3's fit.

You can save the fitted parameter values in your parameter table, either in a new column or in an old one. If the fitted values replace an older set of values, the procedure updates only the parameters that varied in this most recent fit. Moreover, the column heading is updated to include the F-value of the fit (see Figure 10). Thus, you can compare the goodness of two fits by their column headings. In addition, the next column is set to the standard deviation of each fitted parameter.

You can then repeat the whole process, subject to new parameter specifications (e.g., which parameters to vary or which parameter values to use to start the next fit).

```
WOULD YOU LIKE TO SAVE THE FITTED PARAMETER VALUES IN PARAMTAB?Y<
WOULD YOU LIKE TO SAVE THEM IN A NEW COLUMN?Y<
FITTED PARAMETERS SAVED IN COLUMN 2

WOULD YOU LIKE TO FURTHER FIT THE SAME SYSTEM?N<
```

PARAMTAB 4R X 3C

0 PARAMETER NAME	1 INITIAL VALUES	2 FITTED VALUES (F-VAL = 89938.39)	3 STANDARD DEVIATION
K1	1.	1.003954	.002528505
K2	1.	1.001125	.003489576
K3	1.	.5016227	.00318925
A	2.	2.	

Figure 10. Saving the fitted parameter values and leaving FITDIFF.

INTDIFF

Use this entry when you want to numerically integrate your system of differential equations over certain ranges. To use INTDIFF, type the command:

```
CALL INTDIFF <GO>
```

Input

Like FITDIFF, INTDIFF asks for the names of your function and parameter tables; these tables must be organized exactly like those required by FITDIFF. If you do not have a function or parameter table, you can create them within the procedure. INTDIFF then prompts you for the following information:

- the names of the differential equations to include in the system to be integrated
- the name of the independent variable of the system
- the column of your parameter table containing parameter values
- ranges of values for the independent variable over which to perform integration

When INTDIFF prompts you for ranges of your independent variables, you should specify ranges for the integration of your system. Each range consists of a low value, a high value, and an increment. For example, 1.0, 2.0, .1 specifies the range 1.0, 1.1, ..., 1.9, 2.0.

See Figure 11 for details of your interaction with INTDIFF.

```
▶CALL INTDIFF◀

DO YOU HAVE A FUNCTION TABLE?Y◀
NAME OF FUNCTION TABLE?FNTAB◀
WOULD YOU LIKE TO EDIT FNTAB?N◀
DO YOU HAVE A PARAMETER TABLE?Y◀
NAME OF PARAMETER TABLE?PARAMTAB◀
WOULD YOU LIKE TO EDIT PARAMTAB?N◀

AVAILABLE DIFFERENTIAL EQUATIONS TO INCLUDE IN THE NEXT SYSTEM ARE:

Y1 = -K1*Y1

Y2 = K1*Y1-K2*Y2

Y3 = K2*Y2-K3*Y3

LIST OF NAMES OF THOSE DIFFERENTIAL EQUATIONS (SEPARATED BY
COMMAS OR BLANKS) TO INCLUDE
IN THE NEXT SYSTEM?Y1 Y2 Y3◀

NAME OF INDEPENDENT VARIABLE OF SYSTEM?T◀

COLUMN OF TABLE WITH INITIAL PARAMETER VALUES TO
BE USED IN THE NEXT CALL TO MLAB?2◀

ENTER SETS OF RANGES FOR T TO PERFORM INTEGRATION OVER:
(WHEN DONE, HIT <GO> FOR 'LOW VALUE' PROMPT)

        LOW VALUE?0◀
        HIGH VALUE?3◀
        INCREMENT?.05◀

        LOW VALUE?3◀
        HIGH VALUE?5◀
        INCREMENT?.1◀

        LOW VALUE?◀
```

Figure 11. Interactively supplying INTDIFF with input.

Output

INDIFF now calls MLAB to integrate the system. When integration is completed, INTDIFF asks whether you would like to see a graph of all the function values. It also gives you the option of seeing a table of all function and derivative values, which you can then save as a temporary table.

Finally, INTDIFF asks whether you would like to integrate further the same system. If you do, it asks for new independent variable ranges.

Consult Figures 12 and 13 for illustration of INTDIFF's integration results.

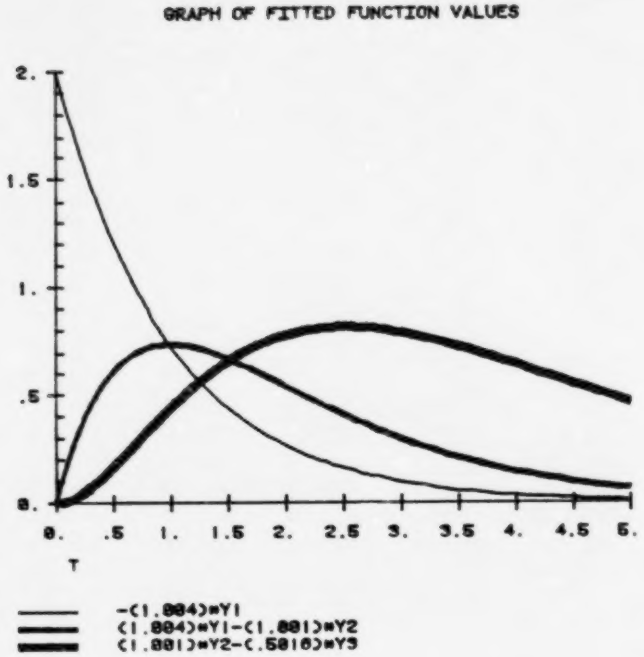


Figure 12. Graph of all of the system's function values.

TABLE OF FUNCTIONS AND DERIVATIVES 62R X 7C

	1 T	2 Y1	3 Y1 DIFF T	4 Y2	5 Y2 DIFF T	6 Y3	7 Y3 DIFF T
1	0.	2.	-2.007987	0.	2.007987	0.	0.
2	.05	1.902068	-1.909566	.09550094	1.813979	.002410605	.09439906
3	.1	1.806945	-1.816097	.1816481	1.634244	.009247731	.1772146
4	.15	1.720363	-1.727164	.2591437	1.467749	.01995797	.2494239
5	.2	1.636150	-1.642625	.3280246	1.31363	.03403425	.311922
6	.25	1.556052	-1.562204	.3906959	1.171066	.05100621	.3655497
7	.3	1.479672	-1.485723	.4459812	1.03932	.07046511	.4110561
8	.35	1.40742	-1.412964	.4947657	.9176415	.09200069	.4491931
9	.4	1.338516	-1.343806	.5376164	.8053864	.1152792	.4805949
10	.45	1.272986	-1.278019	.575456	.7019154	.1399736	.5056695
11	.5	1.210667	-1.215453	.6081191	.6066501	.1658024	.5256331
12	.55	1.151392	-1.155944	.6362376	.5169901	.1924663	.5404063
13	.6	1.095033	-1.099362	.6601817	.436516	.2196039	.5505656
14	.65	1.041425	-1.045542	.6801413	.3646353	.2475179	.556746
15	.7	.9904339	-.9943497	.6966061	.2966997	.2754252	.5592904
16	.75	.9419577	-.9456616	.7096665	.2349945	.3034327	.5584765
17	.8	.8956403	-.8993621	.7201993	.1763724	.3312619	.5546311
18	.85	.8519786	-.8553469	.7276134	.1267146	.3586741	.5486129
19	.9	.8102793	-.8134626	.732912	.07974613	.3861413	.5400394
20	.95	.7706857	-.7736524	.7356216	.0370926	.4126643	.5295376
21	1.	.7326762	-.7357757	.7366645	-.001737764	.4390655	.5172662
22	1.05	.6970873	-.699763	.7356739	-.03673666	.4646199	.5034376
23	1.1	.6626771	-.6654976	.7330477	-.06637467	.4894191	.4883732
24	1.15	.6304271	-.6329196	.7288932	-.09679375	.5134395	.4721604
25	1.2	.5995690	-.6019483	.7233601	-.1222536	.5366443	.455001
26	1.25	.5702063	-.5724627	.7167067	-.1450505	.5589373	.4371375
27	1.3	.542296	-.544442	.7089166	-.1652742	.5803495	.4185997
28	1.35	.5157534	-.5177925	.7001799	-.1831752	.6006213	.3995821
29	1.4	.4904943	-.4924336	.6906397	-.1989633	.6203043	.3802581
30	1.45	.4664866	-.4683329	.6803145	-.2127471	.6386454	.3606207
31	1.5	.4436539	-.4454079	.6693566	-.2247036	.6563929	.3406501
32	1.55	.4219266	-.4235949	.6576746	-.23502	.6729318	.321057
33	1.6	.4012768	-.4028633	.6456799	-.2437433	.6885927	.301236
34	1.65	.3816333	-.3831421	.6334962	-.2510669	.7030779	.2815291

Figure 13. INTDIFF's table of function and derivative values.

FITANDINTDIFF

This entry combines the features of FITDIFF and INTDIFF. To use this entry, type the command:

```
CALL FITANDINTDIFF <GO>
```

After fitting your system, you can obtain integrated function values over any desired range for those fitted parameters. FITANDINTDIFF behaves almost identically to FITDIFF except that it asks you to specify ranges for integrating the system. If you then request a table of fitted functions, the independent variable varies over these specified ranges. You can also obtain a graph of all the integrated functions over the same ranges.

Quiet Entries

To provide greater flexibility, DIFFEQ has quiet entries that do not ask you for any information after you call the procedure. In addition, these entries display no results; rather, the fit results will be returned.

FITDIFFQT

```
CALL FITDIFFQT(function-table, parameter-table,
               data-table, ind-var-col, par-vary,
               par-consts, fit-ans, tab-opts,
               tab-results)
```

FITDIFFQT fits your system quietly. You must supply the nine arguments that comprise the specifications of the desired fit.

Use this entry to perform a fit with minimal interaction (e.g., when you need a fitting subprocedure in a procedure). Once invoked, FITDIFFQT performs the desired fit and returns only those table results determined by the value of tab-opts. The specifications of the arguments are described below.

INTDIFFQT

```
CALL INTDIFFQT(function-table, parameter-table,
               ind-var-name, ranges, int-results)
```

INTDIFFQT quietly integrates your system. It uses the parameter values from column 1 of your parameter table. Once called, it integrates your system and returns the results in int-results. Before the next integration, you can edit your function and parameter tables and specify new ranges for integration. Consult the "Quiet Entry

Arguments" section below for the specifications of the arguments.

FITANDINTDIFFQT

```
CALL FITANDINTDIFFQT(function-table,
                     parameter-table, data-table,
                     ind-var-col, par-vary,
                     par-consts, fit-ans, tab-opts,
                     tab-results, ranges,
                     int-results)
```

FITANDINTDIFFQT is the quiet version of FITANDINTDIFF. Once invoked, it performs the specified fitting and integration of your system without asking you for any information.

Quiet Entry Arguments

The specifications of the arguments required by DIFFEQ's quiet entries are described in detail below.

function-table <table>

This table is identical to that of FITDIFF, except that it must also contain a fourth and a fifth column. Column 4 must contain the corresponding FIXED column of a data table with function values, and column 5 must either be empty or contain any FIXED columns of weights. For a function with no experimental values, the entry in the fourth column should be EMPTY. All quiet entries assume that each function in the table is part of the system. INTDIFFQT does not require a fourth or a fifth column.

On return from FITDIFFQT and FITANDINTDIFFQT, function-table has an additional column containing the texts of the fitted equations.

parameter-table <table>

This table is identical to that of FITDIFF. All quiet entries use the values in column 1 for the initial call to MLAB.

data-table <table>

This table, which is not required by INTDIFFQT, is identical to that of FITDIFF. It must have no EMPTY values.

ind-var-name <text>

This is the name of the independent variable of the system and is required only by INTDIFFQT.

ind-var-col <fixed>	(not required by INTDIFFQT) is the column of the data table with values for the independent variable.
par-vary <text array>	(not required by INTDIFFQT) contains the names of the parameters to vary, e.g., ['A', 'B'], in the initial fit.
par-consts <text array>	(not required by INDIFFQT) contains any constraints to be enforced, e.g., ['A>0', 'B>A'], in the initial fit. If no constraints are to be enforced, it should be EMPTY.
fit-ans <float array>	(not required by INDIFFQT) must be declared before the call. On return, it contains the following fit results of the overall system: [1] number of data points [2] multiple R [3] multiple R^2 [4] standard deviation of regression [5] number of iterations used by MLAB [6] 1 if MLAB converged; 0 if it did not If your system has more than one differential equation, the subsequent elements of fit-ans will be set to the sum of the squares of the residuals for each differential equation.
tab-opts <fixed>	(not required by INTDIFFQT) is 1, 2, or 3, and determines which table results are returned in tab-results. If tab-opts has any other value, no table results are returned.
tab-results <table array>	(not required by INTDIFFQT) must be declared before the call. On return, it contains different table results of the fit, depending on the value of tab-opts.

If tab-opts = 1, then tab-results are as follows:

[1] parameter results table
[2] constraints table (if there are constraints)
[3] analysis of variance table

If tab-opts = 2, then beginning with the fourth element, tab-results contains residuals tables for each differential equation in the system:

[4] residuals table for equation 1
[n+3] residuals table for equation n

If tab-opts = 3, then all of the table results described above are returned.

Remember, if tab-opts has any value other than 1, 2, or 3, <u>NO</u> table results are returned.

ranges
<float array array>

(not required by FITDIFFQT) specifies the ranges for integration performed by MLAB. It must have three columns and a row for each different range. For example, the float array array

```
[[1.0, 2.0, .1],[5.0, 8.0, .2]]
```

specifies the ranges

```
1.0, 1.1, ..., 1.9, 2.0,
```

and

```
5.0, 5.2, ...  7.8, 8.0.
```

int-results
<table>

(not required by FITDIFFQT) must be declared before the call. On return, it is set to a table of the integrated function values and derivatives over the indicated ranges.

DRUGFUN

by Nick Holford
Division of Clinical Pharmacology
University of California at San Francisco

Summary

DRUGFUN uses MLAB to fit the parameters of standard pharmacokinetic models. You must supply a dose-table that specifies both the doses and the names of data-tables containing time and concentration values. The procedure returns the final parameter values and information about the fit in a results-table.

Calling Sequence

```
RAC PUBLIC DRUGFUN <GO>
CALL DRUGFUN <GO>
```

Entry

```
CALL DRUGFUNQT(dose-table,   drugcol,   model,   lag,
                <table>      <fixed>    <text>  <bool>

               weighting,    guesses,      results-name)
                <text>    <float-array>     <text>
```

Citations

Holford NHG. DRUGFUN, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Knott GD. MLAB: A mathematical modelling tool. Computer Programs Biomed 10:271-280, 1979.

Questions or Comments

Send a message to 'HOLFORD' using PROPHET.

Things to Remember

The data-table column containing time values must have a heading starting with TIME as a separate word.

DRUGFUN can fit steady-state models only if the dose and dosing interval are constant.

DRUGFUN is written with the philosophy that pharmacokinetic models can describe pharmacokinetic data, but there is no "best" model that can be selected *a priori*. DRUGFUN enables you to choose the best of a series of models after comparing the fit to each model. You can fit data sets specified in a dose-table to each contending model by repeatedly calling DRUGFUN, changing only the model (or re-estimating lag time) and the initial guesses each time. The procedure returns the results of each fit in a results-table.

The public tables DRUGMODELS and SSDRUGMODELS provide a limited number of models; you can add additional models to these tables to suit your needs. DRUGFUN uses the information in these model-tables to provide a model function, constraints on parameter values, and headings and derivations for the results-table columns.

You must be aware of four kinds of tables: the data-table, the dose-table, the model-tables (DRUGMODELS and SSDRUGMODELS), and the results-table. You must make the data-table before calling DRUGFUN, but you can use the procedure to make the dose-table. DRUGFUN takes the model-tables from the public directory and constructs the results-table to contain the results of the fit.

Data-Table

The concentration-time data to be fit by DRUGFUN must be contained in a data-table. You must store time values in a column whose heading starts with TIME. Other words in the TIME column heading are allowed so long as they follow the word TIME, e.g., TIME (HOURS). If more than one column contains the word TIME as the first word in its heading, DRUGFUN uses the first such column.

You can store concentration values in any other column of the data-table. If you want to fit more than one data-table with the same call to DRUGFUN, make sure that all the concentration columns have the same column number or the same column heading (see the description of drugcol, DRUGFUN's second argument).

DRUGFUN Arguments

DRUGFUN's seven arguments are described below. The interactive version of the procedure prompts you for each argument. The quiet entry, DRUGFUNQT, requires that you supply all seven arguments when you call the procedure.

Dose-table

With a single call to DRUGFUN, you can fit one or more data sets to the same model. The procedure requires a dose-table containing information about each data set within a data-table. The name of each data-table must be a row name in the dose-table. You can define different doses (DOSE column), infusion durations (TAU column), and dosing intervals (DOSINT column) for each data set.

You can make a dose-table as you would any other PROPHET table, or you can create it within DRUGFUN via an interactive quiz. It is important that the column headings of the dose-table be exactly as given above. Only columns suitable for the model to be fit should be included in dose-table. For example, the DOSINT column must not be present if you are fitting a single-dose model. You must always supply the DOSE column. Column sequence is unimportant.

Zero-Order Input Models. If the drug is administered by constant-rate infusion, a zero-order (K0) model is often appropriate. If you know the duration of infusion, you should enter this value in a column headed TAU in dose-table. If you do not know the duration, or if you model drug absorption via a zero-order process, omit the TAU column in dose-table and the procedure will estimate the input duration (TAU) as an additional parameter of the model. If you do not supply an initial estimate for TAU when asked by DRUGFUN (or if you leave the appropriate element of the guesses FLOAT ARRAY empty), the procedure supplies an estimate by using the time of peak concentration in dose-table.

Steady-State Models. Concentration-time data obtained at steady state after multiple doses can be fitted with a steady-state model. You should enter the dosing interval in a column headed DOSINT in the dose-table. The concentration-time data must be related to the time of the last dose. DRUGFUN enables you to estimate lag time at steady state. By following this time convention, you can include concentrations from different dosing intervals in the data-table.

Drugcol

You can indicate the data-table column containing drug concentration values by specifying either the TEXT of the column heading (enclosed in single quotation marks) or the column number. When

DRUGFUN asks for this column, you should type either the column heading (enclosed in single quotation marks) or the column number.

Model

Models that combine a suitable input function and one or more "compartments" describing distribution and elimination can often describe the concentration-time course of a drug in the body after a single dose or at steady state after multiple doses.

The public tables DRUGMODELS (single dose) and SSDRUGMODELS (multiple doses to steady state) contain some standard combined models. (The contents of these tables are shown in Figures 1 and 2.) Input functions can be BOLUS (e.g., rapid intravenous injection), DA (first-order absorption, e.g., oral tablet), or K0 (zero-order or fixed rate, e.g., intravenous infusion).

DRUGMODELS

MODEL NAME	PARAMETERS
BOLUS 1	VOLUME, KEL, LAGTIME
K01	VOLUME, KEL, LAGTIME
KA1	VOLUME, KEL, KA, LAGTIME
BOLUS2	VOLUME, ALPHA, BETA, K21, LAGTIME
K02	VOLUME, ALPHA, BETA, K21, LAGTIME
KA2	VOLUME, ALPHA, BETA, K21, KA, LAGTIME
BOLUS3	VOLUME, ALPHA, BETA, GAMMA, K21, K31, LAGTIME
K03	VOLUME, ALPHA, BETA, GAMMA, K21, K31, LAGTIME
KA3	VOLUME, ALPHA, BETA, GAMMA, K21, K31, KA, LAGTIME

Figure 1. Table of available single-dose drug models.

SSDRUGMODELS

MODEL NAME	PARAMETERS
KA2SS	VOLUME, KEL, KA, LAGTIME

Figure 2. Table of available steady-state drug models.

Models with three compartments are currently available; you can also add more complex models if you need them. Model names are a combination of the input function and number of compartments. For example, BOLUS1 is a rapid input into a one-compartment model; K03 is a zero-order input into a three-compartment model. Steady-state models are distinguished by the suffix SS; for example, KA1SS is a steady-state model for first-order input into a one-compartment model.

Lag

Each model may optionally include another parameter, lag time. Estimation of lag time is often appropriate after oral administration. The procedure includes it in the model if you answer YES to the DRUGFUN question

DO YOU WANT TO ESTIMATE LAG TIME?

or if you specify TRUE in the lag argument of DRUGFUNQT.

Weighting

Ideally, each concentration value should be weighted in inverse proportion to its variance (homoscedasticity assumption). Estimates of this variance are often lacking but can be approximated by some function of the value itself, e.g., the value squared. You can specify a weighting function as any function of Y; the procedure calculates the weights by substituting concentration values for Y in the function you specify. For example, if you had estimates of the concentration variances in column CONCVAR of your data-table, you might use the weighting function 1/CELL('CONCVAR').

Guesses	You must enter an initial estimate for each parameter in response to DRUGFUN prompts. When you call the noninteractive entry DRUGFUNQT, you must supply initial estimates, in parameter name order, in a FLOAT ARRAY. When you use DRUGFUN to fit several data sets with a single call, the procedure uses the same initial estimates. If the parameters for each data set are likely to be quite different, you should make separate calls to DRUGFUN with appropriately different guesses. Figures 1 and 2 display the parameters of each model.
Fitname	You must provide an unused legal table name for the results-table created by DRUGFUN.
Results-table	DRUGFUN outputs a results-table containing information about the parameter estimates and the goodness of the fit(s). Each row in the results-table corresponds to a single data set; the name of the data-table containing that data set serves as the row name. Columns 1 through 8 of the results-table contain the parameter estimates (as well as some derived parameters). These columns' headings are the names of the parameters. The last seven columns of the results-table contain goodness-of-fit information. Their headings and contents are described below.
MULT R**2 Column	This column contains the coefficient of multiple determination. DRUGFUN computes the coefficient of multiple determination from the (weighted) residual sum of squares and (weighted) total sum of squares. If the MLAB fitting procedure fails to find a minimum for the sum of squares, the coefficient may be negative. In this case, the cell is left EMPTY.
SS RESIDUAL Column	This column contains the residual sum of squares. If you have weighted your data, then the sum of squares is the weighted sum of squares (see the Section 42 of the MLAB Manual cited in the summary box).
OBS Column	This column contains the number of valid observations used in the fit. An observation is valid if there are no missing values for the dependent variable, independent variable(s), or weight. Some weighting functions may invalidate observations; for example, "1/Y**2" will cause the weight to be missing if Y = 0.

PARAMETER SE ARRAY Column	MLAB computes the standard error of each parameter estimate and stores the values as a FLOAT ARRAY in the "PARAMETER SE ARRAY" column. Large standard deviations may reflect variability in the data, such as measurement errors, incorrect specification of the model, or high degree of covariance between parameters.
FITTED FUNCTION Column	DRUGFUN substitutes the parameter values for the parameter names in the function definitions. It stores the function(s) as a TEXT ARRAY in the "FITTED FUNCTION" column. The sequence of functions in the array is the same as the sequence of the rows in dose-table.
DATA SET Column	For each data set, the valid observations and associated independent variables, predicted values, and (weighted) residuals are stored as a FLOAT ARRAY ARRAY in the "DATA SET" column. The columns of the array contain the independent variable(s), dependent variable, predicted value, and residual in that order. The rows of the array contain each set of such values for each valid observation. DRUGFUN stores the data from each data set as an element of a FLOAT ARRAY ARRAY ARRAY in dose-table sequence.
DATA TABLE NAME Column	The names of the data-tables specified in column 0 of dose-table are stored as a TEXT ARRAY in the "DATA TABLE NAME" column.

Notes

- DRUGFUN uses a subprocedure UNIFUNSUB to prepare the data for fitting by MLAB. UNIFUNSUB ignores missing data values and checks for errors. If it finds an error, it types an error message and aborts DRUGFUN.
- DRUGFUN derives some conventional pharmacokinetic parameters from the estimated parameters (e.g., clearance, terminal half-life) and adds them to the results-table.
- The model and weighting function that DRUGFUN uses in the fit are shown in the title of the results-table and are also given in the footnotes to this table.

- The procedure imposes logical constraints on parameter values: all parameters must be positive or zero, and hybrid rate constants are ordered ALPHA > BETA > GAMMA.

- Rapidly diminishing exponential terms in the model may cause MLAB to give the warning message:

```
ARITHMETIC UNDER/OVERFLOW
```

This message may be repeated many times and fill the screen. Type CTRL-L to clear the screen as necessary until the fit is completed.

After several screenfuls, you should suspect that your initial parameter estimates were not good. Strange results in the results-table should confirm this suspicion. If you find such results, try again with different initial estimates.

- If the MLAB prompt "*" appears on the screen, something funny has happened in MLAB. Type EXIT followed by <GO> to return to top-level PROPHET.

- DRUGFUN displays the fitting instructions that it sends to MLAB before the fit commences. This allows you to see the exact function being fit, as well as the associated constraints and convergence criteria.

- If a fatal error occurs during DRUGFUN or DRUGFUNQT, the procedure aborts. To enable you to recover from such an error, e.g., when a batch job is running, a BOOL returns when the procedure ends. If an error was detected, the BOOL returns FALSE; if not, it is TRUE.

Related Public Procedures

You can use the results-table with three other public procedures to obtain more information:

- GRAFIT examines the fit and residuals graphically.

- DRUGAUC estimates the area under the concentration-time curve.

- GOFIT shows the parameter estimate, standard errors, and standard deviation of the fit (goodness-of-fit).

You can freely add columns to the results-table, but these other procedures may not work if you change the columns or footnotes provided by DRUGFUN.

References

Derivation of model functions.
Benet LZ. General treatment of linear mammillary models with elimination from any compartment as used in pharmacokinetics. J Pharm Sci 61:536-541, 1972.

Extensive discussion of pharmacokinetic models and methods for obtaining initial estimates.
Wagner J. Fundamentals of clinical pharmacokinetics. Drug Intelligence Publications, Hamilton, Illinois, 1975.

Examples

The accompanying text illustrates how you could use DRUGFUN to fit drug concentration-time data. Suppose you have sampled plasma concentrations after the first dose and later, at steady state, after multiple doses in the same subject.

Figure 3 shows the single-dose time and concentration data recorded in the table DRUG. Figure 9 shows the steady-state data recorded in table DRUGSS. The pharmacokinetic parameters for your experiment are:

volume of distribution = 10 L
KEL = 0.0001/min
KA = 0.05/min

The dosage is 500 mg given every 12 hours (720 min), and bioavailability is assumed to be 1. Plasma concentrations are reported in mg/L.

DRUG 14R X 2C

	1 TIME (MINS)	2 CP (MG/L)
1	5.	8.6
2	10.	19.3
3	20.	41.6
4	30.	36.
5	40.	28.2
6	50.	55.8
7	60.	56.6
8	70.	53.5
9	100.	39.7
10	120.	33.5
11	180.	50.8
12	300.	30.8
13	600.	35.5
14	720.	28.4

Figure 3. Single-dose time and concentration data.

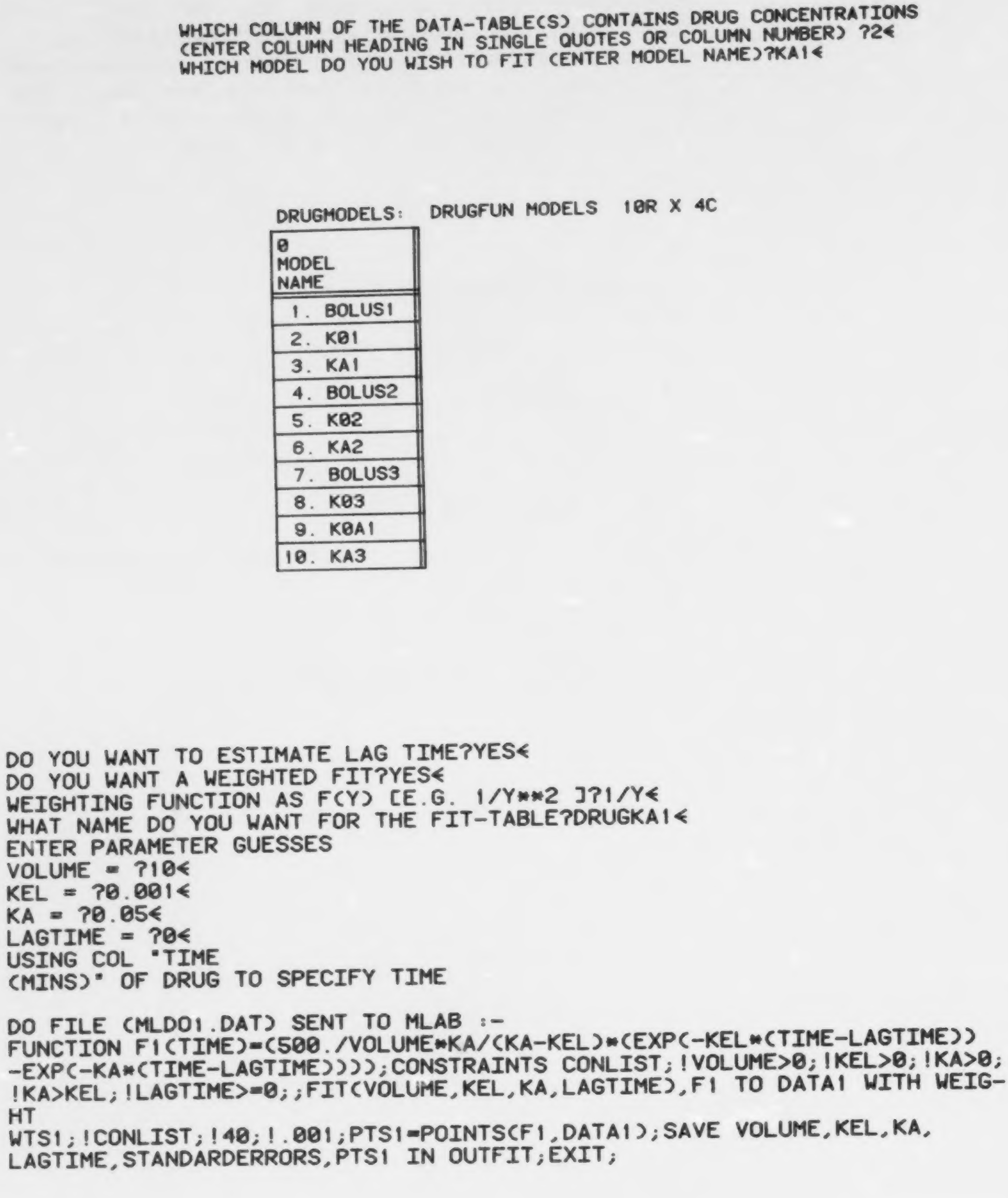
Use DRUGFUN interactively to make a dose-table (DOSE) for the first dose (data-table = DRUG). The procedure obtains a fit using the KA1 model and then estimates and weights the concentrations by their reciprocal (1/Y). Your initial guesses are the parameters used to generate the data. The results are shown in the results-table named DRUGKA1. Consult Figures 4-6 for details.

```
>RAC PUBLIC DRUGFUN<
>CALL DRUGFUN<

DO YOU NEED TO MAKE A DOSE-TABLE?YES<
DO YOU WANT TO MAKE A
      (1) PERMANENT
      (2) TEMPORARY DOSE-TABLE

        (TYPE "1"OR "2")?1<
WHAT NAME DO YOU WANT FOR THE TABLE?DOSE<
WAS DRUG GIVEN BY INFUSION?NO<
SINGLE DOSE OR STEADY STATE -
(TYPE "1" OR "SS")?1<

ENTER NAMES OF DATA TABLES - TYPE "EXIT" WHEN FINISHED

TABLE(1)?DRUG<
DRUG - DOSE = ?500<
TABLE(2)?EXIT<

DO YOU  NEED TO MAKE ALTERATIONS?NO<
DO YOU WISH TO PROCEED WITH FITTING DATA?YES<
```

DOSE 1R X 1C

0 ROWNAME	1 DOSE
1. DRUG	500.

Figure 4. Using DRUGFUN to make a dose-table.



```
WHICH COLUMN OF THE DATA-TABLE(S) CONTAINS DRUG CONCENTRATIONS
(ENTER COLUMN HEADING IN SINGLE QUOTES OR COLUMN NUMBER) ?2<
WHICH MODEL DO YOU WISH TO FIT (ENTER MODEL NAME)?KA1<
```

```
DRUGMODELS:  DRUGFUN MODELS  10R X 4C
+-----------+
|0          |
|MODEL      |
|NAME       |
+-----------+
| 1. BOLUS1 |
| 2. K01    |
| 3. KA1    |
| 4. BOLUS2 |
| 5. K02    |
| 6. KA2    |
| 7. BOLUS3 |
| 8. K03    |
| 9. K0A1   |
|10. KA3    |
+-----------+
```

```
DO YOU WANT TO ESTIMATE LAG TIME?YES<
DO YOU WANT A WEIGHTED FIT?YES<
WEIGHTING FUNCTION AS F(Y) [E.G. 1/Y**2 ]?1/Y<
WHAT NAME DO YOU WANT FOR THE FIT-TABLE?DRUGKA1<
ENTER PARAMETER GUESSES
VOLUME = ?10<
KEL = ?0.001<
KA = ?0.05<
LAGTIME = ?0<
USING COL 'TIME
(MINS)' OF DRUG TO SPECIFY TIME

DO FILE (MLDO1.DAT) SENT TO MLAB :-
FUNCTION F1(TIME)=(500./VOLUME*KA/(KA-KEL)*(EXP(-KEL*(TIME-LAGTIME))
-EXP(-KA*(TIME-LAGTIME))));CONSTRAINTS CONLIST;!VOLUME>0;!KEL>0;!KA>0;
!KA>KEL;!LAGTIME>=0;;FIT(VOLUME,KEL,KA,LAGTIME),F1 TO DATA1 WITH WEIG-
HT
WTS1;!CONLIST;!40;!.001;PTS1=POINTS(F1,DATA1);SAVE VOLUME,KEL,KA,
LAGTIME,STANDARDERRORS,PTS1 IN OUTFIT;EXIT;
```

Figure 5. Using DRUGFUN interactively to fit single-dose concentration-time drug data.

```
DRUGKA1:  FIT TO KA1 (LAG) WITH WEIGHTS (1/Y)  1R X 14C
+--------+----------+------------+----------+----------+------------+------------+
|0       |1         |2           |3         |4         |5  (D)      |6  (D)      |
|DATA    |VOLUME    |KEL         |KA        |LAGTIME   |CLEARANCE   |TERMINAL    |
|TABLE   |          |            |          |          |            |HALF-LIFE   |
+--------+----------+------------+----------+----------+------------+------------+
|1. DRUG |11.10908  |.0006226628 |.07511327 |2.149739  |.006917213  |1113.198    |
+--------+----------+------------+----------+----------+------------+------------+
+--------+-----+--------+-------------+----+---------------+---------------+
|0       |7    |8       |9            |10  |11             |12             |
|DATA    |DOSE |MULT    |SS RESIDUAL  |#   |PARAMETER      |FITTED         |
|TABLE   |     |R**2    |             |OBS |SE ARRAY       |FUNCTION       |
+--------+-----+--------+-------------+----+---------------+---------------+
|1. DRUG |500. |.8338841|677.4437     |14. |[1.293493,.00  |(500./11.10908-|
+--------+-----+--------+-------------+----+---------------+---------------+
+--------+----------------------+-----------+
|0       |13                    |14         |
|DATA    |DATA SET              |DATA TABLE |
|TABLE   |                      |NAME       |
+--------+----------------------+-----------+
|1. DRUG |[[5.,8.6,8.666297,-   |DRUG       |
+--------+----------------------+-----------+

WOULD YOU LIKE TO SEE GRAPHS OF THE FIT(S) AND RESIDUALS?NO<
WOULD YOU LIKE TO ADD AREA UNDER THE CURVE ESTIMATES TO DRUGKA1?NO<
>
```

Figure 6. Results of an interactive fit with the KA1 model.

Using the same dose-table, DRUGFUN can fit a second model to the concentrations in DRUG. This time, use DRUGFUNQT (see Figure 7) with a zero-order input model (K01), a lag time estimate, and the same weighting function. Since the third element of the guesses FLOAT ARRAY (corresponding to the TAU parameter) is empty, the procedure uses the time of the peak concentration (60 min) as an initial estimate for TAU. The results are shown in the results-table DRUGK01 (Figure 8).

```
INITIAL GUESS FOR TAU =  60.

DO FILE (MLDO1.DAT) SENT TO MLAB :-
FUNCTION F1(TIME)=IF (TIME-LAGTIME) > TAU THEN (500./TAU/(VOLUME*KEL)*
(EXP(KEL*TAU)-1)*EXP(-KEL*(TIME-LAGTIME)))ELSE (500./TAU/(VOLUME*KEL)*
(EXP(KEL*(TIME-LAGTIME))-1)*EXP(-KEL*(TIME-LAGTIME)));CONSTRAINTS
CONLIST;!VOLUME>0;!KEL>0;!TAU>0;!LAGTIME>=0;;FIT(VOLUME,KEL,TAU,
LAGTIME),F1 TO DATA1 WITH WEIGHT WTS1;!CONLIST;!40;!.001;PTS1=POINTS
(F1,DATA1);SAVE VOLUME,KEL,TAU,LAGTIME,STANDARDERRORS,PTS1 IN OUTFIT;
EXIT;
>CALL DRUGFUNQT(DOSE,2,'K01',TRUE,'1/Y',[10.0,0.001,,0.0],'DRUGK01')<

USING COL "TIME
(MINS)" OF DRUG TO SPECIFY TIME
```

Figure 7. Using DRUGFUNQT on the same data.

DRUGK01: FIT TO K01 (LAG) WITH WEIGHTS (1/Y) 1R X 14C

0 DATA TABLE	1 VOLUME	2 KEL	3 TAU	4 LAGTIME	5 (D) CLEARANCE	6 (D) TERMINAL HALF-LIFE	7 DOSE	8 MULT R**2	9 SS RESIDUAL	10 # OBS	11 PARAMETER SE ARRAY	12 FITTED FUNCTION	13 DATA SET	14 DATA TABLE NAME
1. DRUG	11.86382	.000472549	19.10954	1.127546	.005606238	1466.826	500.	.8297222	694.4165	14.	[1.141382,.00	IF (TIME-1.127-	[[5.,8.6,8.532612,.	DRUG

Figure 8. Results of a quiet fit with the K01 model.

If you now decide that the KA1 model is the most appropriate, you can fit the steady-state concentrations in data-table DRUGSS. Once again, you use DRUGFUN to make a dose-table (DOSESS), this time specifying a dosing interval of 720 min. Consult Figures 9 through 11 for details.

DRUGSS 14R X 2C

	1 TIME (MINS)	2 CPSS (MG/L)
1	5.	84.4
2	10.	69.3
3	20.	79.7
4	30.	113.
5	40.	85.
6	50.	58.9
7	60.	109.
8	70.	86.9
9	100.	109.
10	120.	95.9
11	180.	105.
12	300.	86.7
13	600.	50.
14	720.	59.

Figure 9. Steady-state time and concentration data.

```
>CALL DRUGFUN<
DO YOU NEED TO MAKE A DOSE-TABLE?YES<
DO YOU WANT TO MAKE A
      (1) PERMANENT
      (2) TEMPORARY DOSE-TABLE

        (TYPE "1"OR "2")?1<
WHAT NAME DO YOU WANT FOR THE TABLE?DOSESS<
WAS DRUG GIVEN BY INFUSION?NO<
SINGLE DOSE OR STEADY STATE -
(TYPE "1" OR "SS")?SS<

ENTER NAMES OF DATA TABLES - TYPE "EXIT" WHEN FINISHED

TABLE(1)?DRUGSS<
DRUGSS - DOSE = ?500<    DOSING INTERVAL = ?720<
TABLE(2)?EXIT< DO YOU  NEED TO MAKE ALTERATIONS?NO<
DO YOU WISH TO PROCEED WITH FITTING DATA?YES<
```

DOSESS 1R X 2C

0 ROWNAME	1 DOSE	2 DOSINT
1. DRUGSS	500.	720.

Figure 10. Using DRUGFUN to make a dose-table.

```
 WHICH COLUMN OF THE DATA-TABLE(S) CONTAINS DRUG CONCENTRATIONS
(ENTER COLUMN HEADING IN SINGLE QUOTES OR COLUMN NUMBER) ?'CPSS
(MG/L)'<

WHICH MODEL DO YOU WISH TO FIT (ENTER MODEL NAME)?KA1SS<




DO YOU WANT TO ESTIMATE LAG TIME?YES<
DO YOU WANT A WEIGHTED FIT?YES<
WEIGHTING FUNCTION AS F(Y) [E.G. 1/Y**2 ]?1/Y<
WHAT NAME DO YOU WANT FOR THE FIT-TABLE?DRUGKA1<
ENTER PARAMETER GUESSES
VOLUME = ?10<
KEL = ?0.001<
KA = ?0.05<
LAGTIME = ?0<
USING COL 'TIME
(MINS)' OF DRUG TO SPECIFY TIME

DO FILE (MLDO1.DAT) SENT TO MLAB :-
FUNCTION F1(TIME)=(500./VOLUME*KA/(KA-KEL)*(EXP(-KEL*(TIME-LAGTIME))
-EXP(-KA*(TIME-LAGTIME))));CONSTRAINTS CONLIST;!VOLUME>0;!KEL>0;!KA>0;
!KA>KEL;!LAGTIME>=0;;FIT(VOLUME,KEL,KA,LAGTIME),F1 TO DATA1 WITH WEIG-
HT
WTS1;!CONLIST;!40;!.001;PTS1=POINTS(F1,DATA1);SAVE VOLUME,KEL,KA,
LAGTIME,STANDARDERRORS,PTS1 IN OUTFIT;EXIT;
```

Figure 11. Using DRUGFUN interactively to fit steady-state concentration-time drug data.

The results of the fit are shown in table DRUGKA1SS (Figure 12). The quality of the fit does not seem to be as good as the single-dose fit if you compare the multiple R-squared values (0.83 versus 0.46). There is not much more you can do with DRUGFUN at this point except try another model. If you need to get better parameter estimates, you must get more data points next time and/or improve your assay for drug concentration.

DRUGKA1SS: FIT TO KA1SS (LAG) WITH WEIGHTS (1/Y) 1R X 15C

0 DATA TABLE	1 VOLUME	2 KEL	3 KA	4 LAGTIME	5 (D) CLEARANCE
1. DRUGSS	9.693611	.00089928	.02128635	7.450581×10^{-7}	.008717271

0 DATA TABLE	6 (D) TERMINAL HALF-LIFE	7 DOSE	8 DOSINT	9 MULT R**2	10 SS RESIDUAL	11 # OBS
1. DRUGSS	770.7802	500.	720.	.4575396	3229.265	14.

0 DATA TABLE	12 PARAMETER SE ARRAY	13 FITTED FUNCTION	14 DATA SET
1. DRUGSS	[5.857253,.00	IF (TIME-7.450-	[[5.,84.4,64.06553,

0 DATA TABLE	15 DATA TABLE NAME
1. DRUGSS	DRUGSS

Figure 12. Results of a steady-state fit with the KA1SS model.

DRUGAUC

by Nick Holford
Division of Clinical Pharmacology
University of California, San Francisco

Summary

DRUGAUC uses a results-table made by the public procedure DRUGFUN to estimate the area under a concentration-time curve. From the area under the curve, it derives drug clearance and steady-state volume of distribution, relying minimally on the pharmacokinetic model used in DRUGFUN.

Calling Sequence

```
RAC PUBLIC DRUGAUC <GO>
CALL DRUGAUC <GO>
```

Entry

```
CALL DRUGAUCQT(results-table, logtrap)
                  <table>     <bool>
```

Citation

Holford NHG. DRUGAUC, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt, Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Send a message to 'HOLFORD' using PROPHET.

Things to Remember

When it estimates clearance and steady-state volume of distribution (VD_{ss}), DRUGAUC assumes that bioavailability is equal to 1.

Clearance and Area Under the Curve (AUC)

Pharmacokinetic models are frequently used to provide estimates of drug clearance. The reliability of these estimates depends heavily on how well the models are suited to the data. Estimates of clearance derived from the area under the concentration-time curve are often preferred, because such estimates are less sensitive to the exact model used. Clearance is derived using the formula:

$$\text{clearance} = \frac{\text{dose} \cdot \text{bioavailability}}{\text{area under the curve}}$$

DRUGAUC assumes that bioavailability is equal to 1. If you know that the bioavailability is less than 1, then you should revise the clearance appropriately.

The linear trapezoidal rule is:

$$AUC_i = \frac{(C_i + C_{i+1})}{2} \cdot (T_{i+1} - T_i)$$

The logarithmic trapezoidal rule is

$$AUC_i = \frac{(C_i - C_{i+1})}{\ln(C_i/C_{i+1})} \cdot (T_{i+1} - T_i)$$

where AUC_i is defined as the area between concentrations C_i and C_{i+1} at times T_i and T_{i+1}. When concentrations are decreasing, AUC_i is estimated more closely by the logarithmic trapezoidal rule. The linear rule is always an approximation to AUC_i, whereas the logarithmic rule is exactly correct during the exponential terminal elimination phase (assuming no errors in the data). DRUGAUC provides two methods of calculating the total (the second of these methods is probably best for general use):

- Linear trapezoidal rule for all data points
- Linear trapezoidal rule when $C_i \leq C_{i+1}$; logarithmic trapezoidal rule when $C_i > C_{i+1}$ (after the maximum observed concentration)

If you obtained the concentration-time data at steady state, DRUGAUC uses only the area under the curve defined by the points. However, after single doses, DRUGAUC must estimate the area under the curve after the last measured concentration by extrapolation. The procedure obtains the extrapolated area from:

$$AUC_{T_{last}-\infty} = \frac{\hat{C}_{last}}{LAMBDA}$$

where $\hat{C}_{last}$ is the predicted value for the last concentration point and LAMBDA is the terminal elimination rate constant. You can usually obtain an estimate of the terminal elimination rate via curve fitting (for example, by using DRUGFUN).

This estimate depends on how well the model is suited to the data. The reliability of the extrapolated portion of the area under the curve must, therefore, be judged by the adequacy of the model. (You can use the public procedure GRAFIT to judge the model's adequacy.)

Clearance estimates from steady-state measurements do not rely on the pharmacokinetic model. The DRUGAUC clearance estimate is essentially independent of the model. If the clearances estimated by DRUGFUN and DRUGAUC differ markedly, then the model that DRUGFUN used was probably inappropriate.

VD_{ss} and Area Under the Moment Curve *AUMC*

Just as you can use the area under the curve to estimate clearance, you can use the area under the moment curve to estimate the apparent volume of distribution at steady state. This volume is defined as the apparent volume required to explain the drug concentration when all tissues in the body have equilibrated at steady state:

$$VD_{ss} = \frac{A_{ss}}{C_{ss}}$$

where A_{ss} is the amount of drug in the body and C_{ss} is the concentration of drug. VD_{ss} is the same as the volume parameter of a one-compartment model, but it is always greater than the volume of the central compartment of multicompartment models.

The area under the moment curve (AUMC) is the integral with respect to time between 0 and infinity of the curve

$$T \cdot f(C, T)$$

where $f(C, T)$ is a function of concentration (C) and time (T). DRUGAUC estimates the AUMC using linear and logarithmic trapezoidal rules similar to those it uses for AUC. The linear trapezoidal rule is:

$$AUMC_i = \frac{(C_i \cdot T_i - C_{i+1} \cdot T_{i+1})}{2} \cdot (T_{i+1} - T_i)$$

The logarithmic trapezoidal rule is:

$$AUMC_i = \frac{(C_i \cdot T_i - C_{i+1} \cdot T_{i+1})}{LAMBDA_i} + \frac{(C_i - C_{i+1})}{(LAMBDA_i)^2}$$

where

$$LAMBDA_i = \frac{\ln(C_i / C_{i+1})}{(T_{i+1} - T_i)}$$

The procedure estimates the AUMC from the time of the last predicted concentration to infinity with the equation

$$AUMC_{C_{last} - \infty} = \frac{\hat{C}_{last} \cdot T_{last}}{LAMBDA} + \frac{\hat{C}_{last}}{(LAMBDA)^2}$$

where LAMBDA is the terminal elimination rate constant estimated by DRUGFUN.

The AUMC depends on the dose, the method of drug administration, clearance, and VD_{ss}. DRUGAUC uses the method of drug administration to calculate the mean input time (MIT) as follows:

Input Model	MIT
bolus	0
first-order	1/KA
zero-order	TAU/2

where KA is the first-order rate constant for drug input and TAU is the duration of a zero-order (constant rate) input. VD_{ss} is then calculated from

$$VD_{ss} = \left(\frac{AUMC \cdot \text{clearance}}{\text{dose} \cdot \text{bioavailability}} - MIT \right) \cdot \text{clearance}$$

DRUGAUC assumes that bioavailability is equal to 1 for this calculation.

For single-dose data, DRUGAUC calculates AUMC using the linear and/or logarithmic trapezoidal rules. DRUGAUC selects the method in exactly the same way that it chooses the AUC method. The total AUMC, percent extrapolated AUMC, and VD_{ss} are added to the results-table. For one-compartment models, VD_{ss} should be the same as the volume parameter.

You can evaluate the appropriateness of a one-compartment model by comparing how well the volume and clearance estimated using the model agree with VD_{ss} and clearance estimated from the area calculations. The AUMC calculation cannot be performed on time-concentration data collected at steady state.

Terminal Half-Life

If you have your own method of estimating the terminal half-life, you may want to use your estimate instead of the value estimated by the model used in DRUGFUN. To use your value, clear the derivation for the TERMINAL HALF-LIFE column in the DRUGFUN results-table, then enter your estimate of terminal half-life into this column. DRUGAUC will now use your estimate to calculate LAMBDA (the elimination rate constant) and will extrapolate the AUC and AUMC from the last measured concentration rather than from the last predicted concentration.

DRUGAUC Arguments

When you call DRUGAUC, the procedure prompts you for the values of the two arguments described below. DRUGAUC's quiet entry, DRUGAUCQT, asks no questions; it requires that you specify these arguments when you call the procedure.

results-table <table>

DRUGAUC requires a results-table made by DRUGFUN. The data tables used by DRUGFUN for the fit need not be available to DRUGAUC; they are contained within the results-table. DRUGAUC adds estimates of the clearance, VD_{ss}, total AUC and AUMC, and percent extrapolated AUC and AUMC to the results-table.

logtrap <bool>

DRUGAUC will use the combined logarithmic and linear methods if logtrap is TRUE, and only the linear method if logtrap is FALSE.

Consult Figures 1 and 2 for illustration of the results of DRUGAUC's calculations for one set of data and the two different values of logtrap.

```
>RAC PUBLIC DRUGAUC<
>CALL DRUGAUC<

NAME OF DRUGFUN FIT-TABLE?DRUGKA1<
DO YOU WANT TO CALCULATE AREA BY

        (1) LOG TRAPEZOIDAL RULE
        (2) LINEAR TRAPEZOIDAL RULE

  TYPE "1" OR "2" ?1<
```

```
DRUGKA1:  FIT TO KA1 (LAG) WITH WEIGHTS (1/Y)  1R X 26C
| 0       || 5  (D)     | 15         | 17       | 18         | 20       |
| DATA    || CLEARANCE  | CLEARANCE  | LOGAUC   | VDSS       | LOGAUMC  |
| TABLE   ||            | (LOGAUC)   | %EXTRAP  | (LOGAUMC)  | %EXTRAP  |
| 1. DRUG || .006914443 | .006902637 | 64.39463 | 11.09236   | 92.47611 |
```

Figure 1. Using DRUGAUC to calculate area via the log trapezoidal rule.

```
>CALL DRUGAUC<

NAME OF DRUGFUN FIT-TABLE?DRUGKA1<

DO YOU WANT TO CALCULATE AREA BY

        (1) LOG TRAPEZOIDAL RULE
        (2) LINEAR TRAPEZOIDAL RULE

  TYPE "1" OR "2" ?2<
```

```
DRUGKA1:  FIT TO KA1 (LAG) WITH WEIGHTS (1/Y)  1R X 26C
| 0       || 5  (D)     | 21         | 23       | 24          | 26       |
| DATA    || CLEARANCE  | CLEARANCE  | TRAPAUC  | VDSS        | TRAPAUMC |
| TABLE   ||            | (TRAPAUC)  | %EXTRAP  | (TRAPAUMC)  | %EXTRAP  |
| 1. DRUG || .006914443 | .006885516 | 64.23491 | 11.04302    | 92.4276  |
```

Figure 2. Using DRUGAUC to calculate area via the linear trapezoidal rule.

Notes

- If DRUGFUN used single-dose models, DRUGAUC assumes that the concentration at time zero is zero (except for BOLUS models -- see below). DRUGAUC calculates the AUC from time zero to the time of the first concentration measurement in the data table and adds it to the AUC derived from the original data.
- If DRUGFUN used a single-dose BOLUS model, DRUGAUC assumes that the initial concentration at time zero is dose/volume, and uses the values for dose and volume that are given in DRUGFUN's results-table. DRUGAUC calculates the AUC from time zero to the time of the first concentration measurement in the original data and adds it to the AUC derived from the original data. This estimate of AUC depends on how well the BOLUS model is suited to the data, because volume estimates depend on the model.

References

Choiu WL. Critical evaluation of the potential error in pharmacokinetic studies of using the linear trapezoidal rule method for the calculation of the area under the plasma level time curve. J Pharmacokinet Biopharm 6:539-546, 1978.

Benet LZ and Galeazzi RL. Noncompartmental determination of the steady state volume of distribution. J Pharm Sci 68:1071-1074, 1979.

MKMODEL

by Nick Holford
Division of Clinical Pharmacology
University of California at San Francisco

Summary

MKMODEL is a modelling procedure that uses MLAB to perform simulation or curve fitting of one or more functions. The functions may be expressed in an integrated or derivative (differential equation) form.

Calling Sequence

```
RAC PUBLIC MKMODEL <GO>
CALL MKMODEL <GO>
```

Entries

```
CALL QMDL <GO>
CALL MKMODELQT(modeltab,  parmspec1,  parmspec2
                <table>     <text>*     <text>*

                constraints,  results-name)
                  <text>         <text>
```

* See the MKMODELQT section for alternatives.

Associated Procedures

(saved in the public table MODELSAVED)

MKMDLTAB
MKPARMTAB
EDITMDL
DATA2TBL
LOOPMDL
FIT2PARM
SETOPTION

Citation

Holford NHG. MKMODEL, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Knott GD. MLAB: A mathematical modelling tool. Computer Programs Biomed 10:271-280, 1979.

Questions or Comments

Send a message to 'HOLFORD' using PROPHET.

Things to Remember

MKMODEL uses a table called MODELOPTIONS. If you do not already have a MODELOPTIONS table, MKMODEL automatically creates one with default values. Make sure the options are set to the values you want.

If a results-table already exists with the name specified as results-name, then the output from MKMODEL is added to the existing results-table as the last row.

Introduction

A useful way to describe systems and test hypotheses is to use simulation to explore potential models of biological or chemical systems and to estimate model parameters from measurements of the system. The public procedure MKMODEL provides the tools to perform both simulation and parameter estimation.

A model consists of one or more functions that can be used for simulation or curve fitting. The term "curve fitting" describes parameter estimation using a set of observations and a mathematical model. Each function consists of a set of parameters and one or more independent variables. Evaluation of the function produces a value for the dependent variable. For convenience, the independent variable is referred to as the X variable; the dependent variable, as the Y variable.

For example, the concentration of drug (C) in the body after an intravenous bolus dose can be described by:

$$C = \frac{DOSE}{VD} \cdot e^{\frac{-CLP \cdot T}{VD}} \qquad (1)$$

where CLP is drug clearance from plasma, VD is volume of distribution, and T is time after the dose. In this example, C is the Y variable and T is the X variable. The parameters are DOSE, CLP, and VD. You could use MKMODEL to simulate this expression for C with different values for DOSE, CLP, and VD over a range of values for T. On the other hand, if you have made measurements of C at known times after a known dose, you could use MKMODEL to estimate the values of the two unknown parameters, CLP and VD, by curve fitting. In this case, DOSE is a parameter constant and CLP and VD are parameter variables.

Models are specified by a model-table containing both the expressions that define the functions, and the source of X values for each function. For curve fitting, the model-table specifies a data-table column containing both X and Y values.

The results of simulation or curve fitting are returned in a results-table. You can use the public procedure GRAFIT to make graphs of the simulated values or fitted and observed values.

If you use MKMODEL for curve fitting, the estimates for the parameter variables and information about the fitting process and the quality of the fit are shown in the same results-table.

Several options for using MKMODEL are stored in a PROPHET table named MODELOPTIONS. For example, the SIMULATION option enables you to use MKMODEL either for simulation or for curve fitting. If SIMULATION is TRUE, then MKMODEL performs a simulation; if it is FALSE, MKMODEL fits a curve. You must have a MODELOPTIONS table available when you use MKMODEL.

Using MKMODEL for the First Time

MKMODEL is actually composed of several subprocedures that are saved separately. The first time you call MKMODEL, these subprocedures are automatically restored and compiled. A message is typed out as each procedure is restored. If you do not have a MODELOPTIONS table, MKMODEL creates a copy of the public MODELOPTIONS table for you. If your MODELOPTIONS table is archived, MKMODEL dearchives it before proceeding.

Figure 1 shows what happens the first time you use MKMODEL. After you RAC MKMODEL from the PUBLIC SAVED table, the procedure creates a MODELOPTIONS table for you. Unless you delete your MODELOPTIONS table, this happens only the very first time you use MKMODEL. Then MKMODEL automatically RAC's all of its subprocedures from the public MODELSAVED table.

```
>RAC PUBLIC MKMODEL<
>CALL MKMODEL<
[MAKING 'MODELOPTIONS' TABLE... DONE]
[RAC ... GRAFIT  GOFIT  MODELTEST MKMDLTAB MKPARMTAB MKFITAB MLSUB
RUNPROPHET RUNMLAB MKPARMFILE MKDATAFILE MKFUNCFILE MKSYSTEM MERGE
GETFUN GETRESULTS GETOPTION SETOPTION ERRORS EDITMDL DATA2TBL
FIT2PARM LOOPMDL GETMGRAPH  DONE]
DO YOU WANT TO EXAMINE MODEL OPTIONS?YES<
```

Figure 1. Calling MKMODEL for the first time.

MODELOPTIONS Table

When you use MKMODEL for a particular problem, you may wish to change some of the default values supplied in the MODELOPTIONS table. The options available are shown below; each option and its value are contained in one row of the table. The defaults are shown in the MODELOPTIONS table displayed in Figure 2.

MODELOPTIONS 12R X 2C

0 OPTION	1 VALUE	2 DATA TYPE
1. ENVIRONMENT	NEWLAB	TEXT
2. SIMULATION	TRUE	BOOL
3. INTERACTIVE	TRUE	BOOL
4. SHOWCMDS	TRUE	BOOL
5. FITCMD	FIT	TEXT
6. ITERATIONS	20	FIXED
7. CONVERGENCE	.001	FLOAT
8. DEQRELERR	.001	FLOAT
9. STDWT	TRUE	BOOL
10. NEWFUNCTION	TRUE	BOOL
11. NEWDATA	TRUE	BOOL
12. TIMES	FALSE	BOOL

Figure 2. MODELOPTIONS Table.

ENVIRONMENT <text>	Set this to the version of MLAB that you want MKMODEL to use. Several versions are available; the most recent is NEWLAB. Earlier versions do not provide as much output and may be less reliable for estimating parameters.
SIMULATION <bool>	Set SIMULATION to TRUE for simulation, FALSE for curve fitting.
INTERACTIVE <bool>	Set this option to TRUE if you want to watch MLAB output during simulation or curve fitting and want to control MLAB's return to PROPHET. (See the "Notes" section below for a description of how to use this option for debugging.)
SHOWCMDS <bool>	Set SHOWCMDS to TRUE if you want PROPHET to display the commands MKMODEL sends to MLAB.
FITCMD <text>	FITCMD controls the display of the results of a fitting. If you want to use MKMODEL interactively, specifying FIT as the value of FITCMD causes the procedure to give you information about the progress of the fit. SILENT FIT gives no messages and is more efficient for noninteractive use.
ITERATIONS <fixed>	Set ITERATIONS to the maximum number of iterations you want MKMODEL to attempt in curve fitting.
CONVERGENCE <float>	This row must contain a convergence factor to be used in MLAB's iterative curve fitting. See the MLAB Manual for a description of this convergence factor.
DEQRELERR <float>	The DEQRELERR model option only affects differential equation defined functions; see the MLAB Manual for a description of its use.
STDWT <bool>	You can standardize the weights used for curve fitting so that the value of their extended product equals 1. MKMODEL standardizes the weights if STDWT is TRUE. A standardized weight W_S is related to its un-standardized value, W, by

$$W_s = W \cdot e^{-LOGWTSUM/ALLOBS}$$

where ALLOBS is the total number of weighted Y values used in curve fitting, and:

$$LOGWTSUM = \sum_{i=1}^{ALLOBS} \log(W_i)$$

NEWFUNCTION
<bool>

If you are simulating or fitting the same model more than once with new parameter values, new data specifications, or new weights (if you have explicitly requested weighting the first time), you can save some CPU time by using the same set of functions.

The value of NEWFUNCTION in MODELOPTIONS is ignored the first time you call MKMODEL in any PROPHET session, and it is assumed to be TRUE. Subsequently, if NEWFUNCTION is FALSE, the procedure reuses the function specification and fitting commands from the last call. Until NEWFUNCTION is changed to TRUE, MKMODEL ignores any changes in model-table or MODELOPTIONS that would affect the function specification or fitting commands.

[Use NEWFUNCTION = FALSE with care!]

NEWDATA
<bool>

In the same way that you can avoid respecifying functions by setting NEWFUNCTION to FALSE, you can reuse the same set of data and weights for repeated calls to MKMODEL by setting NEWDATA to FALSE. The first time you use MKMODEL in any PROPHET session, the procedure ignores the value of NEWDATA and assumes it is TRUE.

TIMES
<bool>

Set TIMES to TRUE if you want MKMODEL to report the number of CPU seconds spent in various parts of the subprocedure MLSUB.

After MKMODEL RAC's all of its subprocedures, it displays the MODELOPTIONS table and gives you the opportunity to change options (see Figure 3). In the example below, the SIMULATION option is changed to FALSE. You are now ready to perform curve fitting, an example of which is shown at the end of this description.

Note that MKMODEL displays MODELOPTIONS only the first time you RAC the procedure, unless you delete the table. Therefore, before you call MKMODEL, you should display MODELOPTIONS to be sure the options are appropriate for what you want to do. Once you RAC MKMODEL, you can change options by using the SETOPTION subprocedure.

SETOPTION Subprocedure

The SETOPTION subprocedure enables you to change option values in MODELOPTIONS. When you give the command

```
CALL SETOPTION <GO>
```

the procedure prompts you for the name of the option you wish to change. When you respond, you need only type enough characters to uniquely identify the option.

Typing DIS will display the MODELOPTIONS table with any changes that have been made. The procedure then prompts you for the new value of the option you specified. No surrounding quotes are required for values whose data type is TEXT (e.g., entering NEWLAB for the environment option).

When you finish changing options, type EXIT or press <GO> in response to the prompt,

```
OPTION?
```

```
DO YOU WANT TO CHANGE MODEL OPTIONS?YES<
TYPE 'EXIT' WHEN FINISHED
OPTION?SIMULATION<
VALUE ?FALSE<
OPTION?EXIT<
```

Figure 3. Changing the SIMULATION option in the MODELOPTIONS table.

Model-Table

After the first call to MKMODEL, you can make a model-table either by calling MKMODEL again or by issuing the command:

```
CALL MKMDLTAB <GO>
```

Each row of the model-table contains all the information needed for a single function. Both MKMODEL and MKMDLTAB prompt you for the contents of each column. The heading, data type, and contents of each column are as follows:

NAME Column
<text>

Use this column to specify a name for each function. If you give no names, MKMODEL creates them: F1 for the function in row 1, F2 for the function in row 2, and so on. A function's name identifies it if you use it as a subfunction (see the "Notes" section below).

ARGUMENT Column
<text>

This column must contain an argument list specifying the name of the X variable(s) in each function defined by an argument list. If you use a subfunction, the argument list may also contain the names of parameters you want to pass to the subfunction.

FUNCTION Column
<text>

For each function, this column must contain an expression describing the function in terms of parameters and one or more X variables. The parameter names must be the same as the names in the parameter-table (see the "Parameter-Table" section below), and the X-variable names must be the same as the names specified in the ARGUMENT column of the model-table.

Because MKMODEL invokes MLAB, the function expression may contain any of the MLAB features, such as ROOT or INTEGRAL, that are not available in PROPHET. See the *MLAB Manual* for details.

X BOUND Column
<float>

Functions defined by their derivatives (differential equations) can be evaluated only if you specify the initial X and Y values in the X BOUND and Y BOUND columns. The X bound must not be within the range of X values being simulated or fitted, but may be equal to the first or last value in the range. You must leave this column empty if you are not using differential equations.

Y BOUND Column
<float> or <text>

This column contains the value of Y corresponding to X bound in the X BOUND column. It may be either a constant or an expression involving one or more parameters. In the latter case, you must specify it as a TEXT. For example, you could write equation 1 in derivative form as:

$$\frac{dC}{dT} = \frac{-CLP}{VD} \cdot C$$

At time T = 0, when the dose is first given, the value of C is the dose divided by the volume of distribution. Therefore, the value of X BOUND is 0 when Y BOUND is DOSE/VD. In this example, the values of the X variable may extend from 0 to any positive value.

DATA Column
<text>

You must provide the values of the X variable(s) required for curve fitting in a separate data-table. The name of the data-table must be contained in the DATA column.

For simulation, you can use a table containing the X-variable values you want to use and specify the values in the same way as for curve fitting. However, you may sometimes want to specify a range of X values without making a table to contain them. Since MKMODEL invokes MLAB to perform a simulation or a fit, you can specify a range of X values by using the MLAB LIST command within the DATA column of model-table.

For example, to specify the X values 1, 3, 5, 8, and 10, you would use:

```
LIST (1, 3, 5, 8, 10) <GO>
```

Notice that you must provide the command word LIST as well as the parentheses. To specify a range of X values from 0 to 10 with increments of 2, you would use:

```
LIST (0:10:2) <GO>
```

You can mix these formats quite freely. For example,

```
LIST (0:10:2, 15, 20, 25, 30:100:10) <GO>
```

produces the set of values 0, 2, 4, 6, 8, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100. If you have more than one X variable in the function, you can produce two or more lists by MLAB concatenation. For example,

```
LIST (0:10:2) &' LIST (1:6:1) <GO>
```

produces a two-column matrix with pairs of X values 0,1; 2,2; 4,3; 6,4; 8,5; 10,6. Consult the MLAB Manual for complete descriptions of the concatenation operator (&') and the LIST command.

Subfunctions do not require an entry in the DATA column. When you are prompted for DATA, type CTRL-E to indicate the use of a subfunction.

X Column
<fixed> or <text>
(or an array of
one these types)

If the DATA column of the model-table holds the name of a data-table containing your X values, then this column should contain the column number (FIXED) or column heading (TEXT) of the data-table column containing the X values. If the specified data-table contains the values of more than one X variable, input a FIXED ARRAY or TEXT ARRAY to indicate the data-table columns you want to use.

Note: THE NEXT THREE COLUMNS ARE REQUIRED FOR CURVE FITTING ONLY. Their contents do not affect the use of MKMODEL for simulation.

Y Column
<fixed> or <text>

This column contains the number or heading of the data-table column containing Y-variable values.

WEIGHT Column
<text>

Weighting of observations is frequently desirable for curve fitting. The object is to correct for differences in the typical error of each observation. Ideally, the correction is made by using the reciprocal of the variance, but often this is not known and is derived in some way from the actual observations. If errors are proportional to the size of the observation (Y), then the square of the observation (Y**2) is an estimate of its variance and the weight should be 1/Y**2.

You must specify weights with a weighting function, which must follow the rules of column derivations, with one exception: the occurrence of the letter Y in the function is taken to mean the value of the dependent variable. Thus, a weighting function of 1/Y**2 would have the same effect as 1/CELL(2)**2 when the Y values are in column 2 of the data table. If you have some other estimate of weights in the data-table (e.g., in column 3), you could use these weight values by specifying a weighting function of CELL(3) or CELL('WEIGHT'), if the heading of column 3 is WEIGHT.

When you do not specify a weighting function in the WEIGHT column of model-table, the procedure assigns a default weight of 1.

SCALE Column
<float>

If you are fitting more than one set of data simultaneously, you can weight the Y values in each set by a constant factor supplied as a FLOAT value in this column.

You should choose the value of this scale factor, S, to make E*SQRT(S) the same for each data set, where E is the standard deviation of the data set. For example, suppose that data set A has a standard deviation of 1 and data set B has a standard deviation of 10. Using a scale factor of 1 for A, the scale factor for B should be 0.01. The default SCALE value is 1.

Figure 4 shows how, once you have called MKMODEL, you could make a model-table to be used for curve fitting. MKMDLTAB's interactive quiz would proceed identically if you called MKMDLTAB before calling MKMODEL.

```
DO YOU HAVE A MODEL TABLE?NO<
DO YOU WANT TO USE DIFFERENTIAL EQUATIONS?YES<

INDEPENDENT VARIABLE NAME?T<

PLEASE SUPPLY THE FOLLOWING FOR EACH FUNCTION:
        1) NAME
        2) DEF  - DERIVATIVE OF FUNCTION
        3) XBND - X BOUNDARY VALUE
        4) YBND - Y BOUNDARY VALUE
        5) DATA - DATA TABLE

         [TYPE "EXIT" WHEN FINISHED]


FUNCTION (1):

NAME?PLASMA<
DEF?(KA*GUT - CLP*PLASMA)/VD<
X BOUNDARY?0<
Y BOUNDARY?0<
DATA?PEANUT<
"T" COL NUMBER?1<
Y COL NUMBER?2<
WEIGHTING FUNCTION?E^<
SCALING FACTOR?E^<

FUNCTION (2):

NAME?GUT<
DEF?-KA*GUT<
X BOUNDARY?0<
Y BOUNDARY?DOSE<
DATA?E^<

FUNCTION (3):

NAME?EXIT<

WHAT NAME DO YOU WANT FOR THE MODEL FUNCTION TABLE?DEQMDL<
[MAKING MODEL TABLE "DEQMDL"... DONE]

DO YOU WANT TO EDIT THE MODEL FUNCTIONS?NO<
```

DEQMDL 2R X 10C

0 NAME	1 ARGUMENT	2 DEFINITION	3 X BOUND	4 Y BOUND	5 DATA	6 X
1. PLASMA	T	(KA*GUT - CLP*PLASMA)/VD	0.	0	PEANUT	1
2. GUT	T	-KA*GUT	0.	DOSE		

0 NAME	7 Y	8 ERROR	9 WEIGHT	10 SCALE
1. PLASMA	2			
2. GUT				

Figure 4. Making a model-table for curve-fitting.

Parameter-Table

You must specify the names and values for all parameters in the model in a parameter-table. Column 0 (ROWNAME column) should contain the parameter names, and column 1 should contain the parameter values. If you are performing curve fitting, the parameter values are used as initial estimates. Otherwise, they are the parameter values for the simulation. You can add other columns to the parameter-table if you wish. For example, column 3 could contain the units of each parameter, and column 4 should contain a brief description of each parameter. MKMODEL ignores all parameter-table columns except column 0 and column 1.

After you first call MKMODEL, you can make a parameter-table either by calling MKMODEL again or by using the command:

```
CALL MKPARMTAB <GO>
```

Both MKMODEL and MKPARMTAB prompt you for all the required information, as is shown in Figure 5.

```
DO YOU HAVE A PARAMETER TABLE?NO<

ENTER PARAMETER NAMES AND VALUES -
 TYPE 'EXIT' WHEN FINISHED

       NAME?DOSE<      DOSE =?1000<
       NAME?CLP<       CLP =?.15<
       NAME?VD<        VD =?8<
       NAME?KA<        KA =?.2<
       NAME?EXIT<

WHAT NAME DO YOU WANT FOR THE MODEL PARAMETER TABLE?PARMS<
[MAKING TABLE 'PARMS'... DONE]


                      PARMS   4R X 1C
                     ┌───────┬┬──────┐
                     │0      ││1     │
                     │NAME   ││VALUE │
                     ├───────┼┼──────┤
                     │1. DOSE││1000. │
                     │2. CLP ││.15   │
                     │3. VD  ││8.    │
                     │4. KA  ││.2    │
                     └───────┴┴──────┘
```

Figure 5. Making a parameter-table after calling MKMODEL.

Results-Table	The contents of the results-table depend on whether you use MKMODEL for simulation or for curve fitting. Each row of the results-table corresponds to a single call to MKMODEL. The procedure uses the same results-table if you give the same results-name; otherwise, it creates a separate, one-row results-table with each call. The results-table made by MKMODEL contains information about the parameter estimates and the goodness-of-fit. The first columns contain the parameter estimates; their headings are the names of the parameters used in the function(s). The remaining columns are arranged in the manner described below:
MULT R**2 Column	MKMODEL returns the coefficient of multiple determination in this column. It is computed from the unweighted residual sum of squares and the unweighted total sum of squares.
LOG L Column	This column contains the log likelihood for the model using the final parameter values. This value is independent of the weighting scheme, so you can use it to compare the fits from different models.
SS RESIDUAL Column	The procedure returns the residual sum of squares in this column. If you weight your data, the sum of squares is the weighted sum of squares.
#OBS Column	This column contains the number of valid observations used in the fit. An observation is valid if there are no missing values for the dependent variable, independent variable(s), or weight. Some weighting functions may invalidate observations; for example, 1/Y causes a missing weight if Y = 0.
NOTES Column	This column contains the parameter names and values used in the simulation. If you use MKMODEL for curve fitting, the occurrence of convergence is shown (with the number of iterations used in parentheses). If there were any active constraints, these are shown followed by the value of the LaGrange multipliers. If standardized weights were used, this is indicated by STDWT.

PARAMETER SE ARRAY Column	At the end of the fit, the procedure computes the standard error of each parameter estimate and stores the values as a FLOAT ARRAY in this column. Large standard errors may reflect measurement errors, poor specification of the model, or a high degree of covariance between parameters.
COVARIANCE MATRIX Column	This column contains the covariance matrix for the parameters.
FUNCTION Column	The fitted parameter values are substituted into the function definition(s) in place of the parameter names. MKMODEL then stores the function or functions as a TEXT ARRAY in this column. The sequence of functions in the array is the same as the sequence of the model-table rows.
DATA SET Column	The valid observations, associated independent variables, predicted values, and (weighted) residuals are stored in a FLOAT ARRAY ARRAY for each data set. The columns of each such array contain the independent variable(s), dependent variable, predicted value, and residual, in that order. The rows of one these arrays contain each set of such values for each valid observation. MKMODEL stores the array of data from each data-set as an element of a FLOAT ARRAY ARRAY ARRAY in this column. The sequence of the data sets is the same as in model-table.
DATA TABLE NAME Column	MKMODEL stores the names of the data-tables specified in column 5 of model-table as a TEXT ARRAY in this column.

If you use MKMODEL for simulation, the last part of the results-table (following the parameter estimates) contains only the NOTES, DATA SET, and DATA TABLE NAME columns. The DATA SET column contains a FLOAT ARRAY ARRAY for each function. This has the X and Y values produced by the simulation. Of course, there are no observed or residual values, as there are in the results from curve fitting.

Figure 6 shows a typical results-table produced by MKMODEL after curve fitting.

PEANUTFIT 1R X 13C

0 MODEL TABLE	1 CLP	2 VD	3 KA	4 MULT R**2	5 LOG L	6 SS RESIDUAL
1. DEQMDL	.1784814	8.858464	.2203716	.8646616	-22.38944	612.1808

0 MODEL TABLE	7 # OBS	8 NOTES	9 PARAMETER SE ARRAY	10 COVARIANCE MATRIX
1. DEQMDL	6.	CONVERGED (3),-	[.06195828,1.	[[1.881223E-5,-.000

0 MODEL TABLE	11 FUNCTION	12 DATA SET	13 DATA TABLE NAME
1. DEQMDL	[(.2203716*G	[[[1.,24.,22.09649,1.9035	[PEANUT]

Figure 6. Typical curve-fitting results-table.

Quiet Entry MKMODELQT

The quiet entry to MKMODEL is called MKMODELQT. Use the command:

```
CALL MKMODELQT(modeltab, parmspec1, parmspec2,
                constraints, results-name)
```

The arguments are described below.

MODELTAB <table>

This argument must be the name of a model-table, as described above.

PARMSPEC1 and PARMSPEC2

You can specify these arguments in four possible ways. Table 1 shows the ways to specify parmspec1 and Table 2 shows the ways to specify parmspec2.

Table 1. Ways to Specify Parmspec1

	Type	Value
1.	<table>	Parameter-table
2.	<text>	Parameter list or variable list*
3.	<text array>	1. Parameter list 2. Variable list*
4.	<text array array>	1. Parameter name array 2. Variable name array

Table 2. Ways to Specify Parmspec2

	Type	Value
1.	<text>	Variable list*
2.	<float array>	Parameter values
3.	<float array>	Parameter values
4.	<float array>	Parameter values

*Curve fitting only.

In general, the simplest way to specify the parameter names, their values, and the parameter variables to be estimated is to use a parameter-table for parmspec1 and list the parameter variables in parmspec2. However, it is not always necessary to make a parameter-table, because you can specify the names and values in a variety of ways. Use of the word 'list' in Table 1 refers to a TEXT containing the names separated by commas or spaces. 'Parameter list' refers to all parameters in the model; 'variable list', to parameters whose values are to be estimated.

You can combine the parameter list and the variable list in the same TEXT by using a slash (/) to separate them. If you do not use a slash, or if there are no names after the slash, MKMODEL assumes that all parameters are parameter variables. For example, to specify the parameters in equation 1 (DOSE, CLP, VD) and to estimate the values of the parameter variables CLP and VD, you would give the TEXT expression 'DOSE, CLP, VD/CLP, VD' as the value of parmspec1.

If you use a FLOAT ARRAY for parmspec2, the sequence of parameter values must correspond exactly to the parameter names in parmspec1. If you use a variable list for parmspec2, the sequence of names in the list need not be the same as the parameter list. *If you leave a parameter value empty, MKMODEL uses a random number between 0 and 1.*

CONSTRAINTS
<text> or
<text array>

You may supply one or more constraints to control the parameter values during curve fitting. (See the *MLAB Manual* for a description of constraints). You can specify a constraint list as a TEXT, with the constraints separated by commas or spaces. (Do not use spaces within a single constraint.) Alternatively, you can use a TEXT ARRAY to specify the constraints.

RESULTS-NAME
<text>

You must supply a legal PROPHET table name for this argument; MKMODEL uses results-name to make a table containing the results. If the table with the name you supply in results-name already exists, MKMODEL adds the results to the existing table. When necessary, the procedure adds new columns to the results-table so that the results of curve fitting can be added to a results-table made after simulation.

Quick
Interactive
Entry, QMDL

There is a quick interactive entry to MKMODEL called QMDL. QMDL gives short prompts for the arguments required by MKMODELQT. If you supply a parameter table for parmspec1, QMDL gives no prompt for parmspec2 when you use the procedure for simulation. QMDL requires no quotation marks around the results-table name. However, you must use single quotation marks to indicate the use of a TEXT constant for parmspec1, parmspec2, or constraints.

Notes

Subfunctions

Subfunctions are a convenient way to express the main function in a compact form. For example, plasma drug concentrations after an oral dose can be described by the expression

$$PLASMA = \frac{DOSE \cdot KA}{VD \cdot (KA - CLP/VD)} \cdot \left[e^{\frac{-CLP \cdot T}{VD}} - e^{-KA \cdot T} \right]$$

where KA is a first-order absorption rate constant, and CLP, VD, and T are as defined above. You can write PLASMA as a main function, using the subfunction, E, in two places:

```
PLASMA =
  DOSE * KA * (E(T, CLP/VD, KA) + E(T, KA, CLP/VD))
```

The argument for PLASMA is T, which should appear in the ARGUMENT column of model-table.

```
     E = EXP((-K1 * T)/(VD * (K2 - K1)))
```

T, K1, and K2 are arguments of the subfunction E, and should be shown as T, K1, K2 in the ARGUMENT column of model-table.

MLSUB

MKMODEL is primarily designed for modelling a wide variety of functions using a standardized input-and-output format based on tables. The heart of this procedure is another procedure named MLSUB. MLSUB has no table arguments, a fact you should consider if you want to develop special-purpose modelling procedures. MLSUB returns its output in an array of type ANY. All the items provided in the MKMODEL results-table derive from this array, as do some items that are returned but are not included in the table.

MLSUB calls three subprocedures, MKPARMFILE, MKDATAFILE, and MKFUNCFILE, before calling MLAB to perform the fit. If NEWFUNCTION is FALSE in the MODELOPTIONS table, then MKFUNCFILE is not called. If the NEWDATA option is FALSE, MKDATAFILE is not called. MLSUB calls the GETRESULTS subprocedure to retrieve values from MLAB, calculate the total sum of squares and log-likelihood, and tidy up the data arrays if you have used zero weights.

Restriction

Because of an MLAB restriction, you cannot mix integrated and derivative functions in the same model for curve fitting.

Speed of Parameter Estimation

Parameter estimation using derivative functions tends to be slower than estimation using the integrated form of the same model. The fitting process is less tolerant of initial parameter guesses that are very different from the optimal value. If you are fitting more than one set of data at the same time, MLAB requires that all the data have identical values, sorted by X, for the X variable. MKMODEL sorts and merges your data sets to overcome this restriction. You pay a price in terms of increased computation time, because MKMODEL must create dummy observations with zero weights.

MLGLOBAL Array

Communication between MLSUB's different subprocedures depends on critical values stored in an ANY ARRAY named MLGLOBAL. MLGLOBAL is a global variable and does not disappear when MKMODEL finishes. If you set the NEWFUNCTION option in the MODELOPTIONS table to FALSE, subsequent calls to MKMODEL rely on some of the values stored in MLGLOBAL. Alter it with care.

Debugging

The INTERACTIVE model option is useful for debugging purposes. If MLAB aborts during simulation or curve fitting, the only indication you receive is the appearance of the MLAB prompt, *. If you see this prompt, return to PROPHET by typing

```
*EXIT <GO>
```

Ignore the ensuing error message, and run MKMODEL again with the INTERACTIVE option set to TRUE.

After you give the DO MLABGO command (see Figure 7), you receive an error message before MLAB aborts. NEWLAB provides the most informative messages. If you get the error message

```
NO SPACE
```

you are using a function or functions that are too large. In most cases, you can overcome this problem by using subfunctions (see the note on subfunctions above). The MLAB Manual may give you some clues for deciphering error messages. The most likely cause for an error is an invalid function expression. You can look at each of your functions by using the command

```
TYPE function-name
```

where function-name is the name you gave the function. If you do not see the function typed out, you know there is a mistake in your definition of the function.

Restoring Disk-Saved Items

MLAB has two basic commands for restoring data and files saved on disk. The USE command restores data items, and the DO command starts execution of commands saved in a DO FILE. When MKMODEL invokes MLAB, the command DO MLABGO causes the MLABGO DO FILE to be read from disk, and the following three commands are executed automatically:

```
USE MLDATA (restores data derived from PROPHET
                data-tables)

USE MLPARM (restores parameter values)

DO MLFIT    (executes the MLFIT DO FILE, which
                contains function definitions and
                instructions for simulation or
                curve fitting)
```

MKMODEL's Commands to MLAB

If you want to review the commands given to MLAB after MKMODEL has finished, you can access them with the command

```
text-var = MLGLOBAL[20] <GO>
```

where text-var is the name of a TEXT variable that will contain the commands. After giving this command, you can manipulate text-var like any other TEXT variable.

Associated Procedures in MODELSAVED

EDITMDL

If you make a mistake in your function definition or wish to change the definition, use the command:

```
CALL EDITMDL <GO>
```

EDITMDL prompts you for the name of a model-table and then transfers the function name, argument list, and function definition to the PROPHET text editor. You may change any of the parts of one or all of your functions. After editing, type

```
EX <GO>
```

to save the edited functions in the model-table. EDITMDL makes it easy to test different models with the same data-table specifications.

If you would like a top-level copy of your function specifications in the format used in the editor, use the MDL2TEXT command. This procedure prompts you for a model-table name and then for a name for the top-level copy of function specifications.

DATA2TBL

If you want to create a table containing the Y values generated by simulation or predicted by curve fitting, use the command:

```
CALL DATA2TBL <GO>
```

These Y values are stored in the DATA SET column of a results-table, along with the X value(s) used to generate them. After curve fitting, the observed Y values and weighted residuals are also stored in this column.

Once called, DATA2TBL prompts you for the name of the results-table. If the results-table contains more than one row, the procedure asks you for the number of the row that you wish to transfer; if there is more than one data set, DATA2TBL asks for the data set number corresponding to the model-table row number used to generate the data set. You are then prompted for the name of a new table to contain the data set. The columns of the new table are shown in Table 3.

Table 3. Columns of New Data Table Produced by DATA2TBL.

Column Number	Column Heading	Contents
1	X	First X-variable values
2 - N	X2 - XN	Values of X variables 2 through N
N + 1	Y	Predicted Y values from simulation OR observed Y values used for curve fitting
N + 2	PREDICTED	Predicted Y values from curve fitting
N + 3	RESIDUAL	Weighted residuals (YOBS-YPRED)* WEIGHT

(N denotes the number of independent variables.)

The title of the new table contains the value of the NOTES column for the selected data set.

FIT2PARM	After curve fitting, you may wish to use the new parameter estimates in a parameter-table for further curve fitting or simulation. FIT2PARM provides a simple means of transferring one or more parameter values from a curve-fitting results-table to a parameter-table. When you give the command

```
CALL FIT2PARM <GO>
```

the procedure prompts you for the name of a results-table. If the table you name contains more than one row, FIT2PARM asks you to indicate the row number of the results-table containing the parameter(s) to be transferred. It then prompts you for the name of a parameter-table to receive the parameter values. Finally, you must specify the names of the parameters as a list separated by spaces or commas, or simplv press <GO> to transfer all the parameter values from the results-table.

LOOPMDL	To use MKMODEL with a series of different data-tables and the same model and parameter specifications, call the LOOPMDL procedure:

```
CALL LOOPMDL(modeltab, parmspec1, parmspec2,
             constraints, result-name,
             namelist)
```

The arguments of LOOPMDL are identical to those of MKMODELQT, with the addition of the final argument, namelist. Namelist is a TEXT ARRAY specifying the names of one or more data-tables.

Once called, LOOPMDL takes the first name in namelist, puts it into the DATA column in the model-table, and calls MKMODELQT. It then repeats this process for every other data-table named in namelist. The X and Y value columns in these data-tables are defined by the X and Y columns in the model-table.

If you RAC LOOPMDL before you RAC MKMODEL, your first call to LOOPMDL will automatically RAC MKMODEL and its associated subprocedures.

You can use LOOPMDL for both simulation and curve fitting.

Batch Operation

Curve fitting and simulation sometimes require extensive computation, so you may wish to use MKMODEL in a batch job to avoid waiting for the procedure to run while you are at the terminal or to reduce computing costs by running the job at night. To use MKMODEL most efficiently in batch mode,

- Use the following MODELOPTIONS values:

```
INTERACTIVE  = FALSE
SHOWCMDS     = FALSE
FITCMD       = 'SILENT FIT'
TIMES        = FALSE
```

- Start your batch job with the commands:

```
RAC PUBLIC MKMODEL <GO>
```

  or

```
ACCESS PUBLIC MODELSAVED <GO>
RAC LOOPMDL FROM MODELSAVED <GO>
```

- Call MKMODELQT or LOOPMDL as described above.

This method is completely quiet. It asks no questions during the subsequent computations.

Whenever you RAC the MKMODEL group of procedures during a PROPHET session, the procedure displays the MODELOPTIONS table and asks whether you want to change any options. This does not occur during batch jobs; MKMODEL does not automatically dearchive MODELOPTIONS in batch jobs.

If you want to keep a separate table of options to use only during batch procedures, make a table called BATCHOPTIONS from your MODELOPTIONS table containing the option values you want to use in batch. Then, before calling MKMODEL in batch, give the command:

```
MODELOPTIONS = BATCHOPTIONS <GO>
```

References

MLAB Beginner's Guide (May 1979) Division of Computer Research and Technology, NIH, Bethesda, MD.

MLAB Reference Manual (April 1980), 9th ed. Division of Computer Research and Technology, NIH, Bethesda, MD.

MLAB Applications Manual (January 1981), 3rd ed. Division of Computer Research and Technology, NIH, Bethesda, MD.

Examples

The use of MKMODEL is illustrated by the one-compartment pharmacokinetic model with first-order absorption described above. You can express this model as a system of differential equations, a single integrated equation, or a system of integrated equations using subfunctions.

In the PROPHET Primer you will find a description of Dr. Althea Lerner's investigations of the absorption and elimination of peanut oil in elephants. The blood concentration of peanut oil in the elephant Chris was measured on six occasions after a heavy meal of peanuts. The usual amount of peanut oil ingested during such a meal is 1000 mg/Kg (A. Lerner, personal communication). The parameters of the model to describe the peanut oil concentrations (mg/L) are the dose (mg/Kg), plasma clearance (CLP, L/min/Kg), volume of distribution (VD, L/Kg) and absorption rate constant (KA, min-1). The parameters to be estimated are CLP, VD, and KA, and initial estimates were obtained from inspection of a graph of the data.

The first example shows how you can make a model-table (DEQMDL) for a system of differential equations and a parameter-table (PARMS) to store the names and initial values. The value of SIMULATION in the MODELOPTIONS table is set to FALSE before you call MKMODEL, so the procedure continues by obtaining the names of the parameters to be estimated and the constraints (none required). The results will be stored in a table named PEANUTFIT. The SHOWCMDS and INTERACTIVE options are TRUE, so the instructions to MLAB are typed out and curve fitting starts when you give the DO MLABGO command.

The next example shows the use of MKMODEL to fit the same data using a single integrated equation. A new model-table, INTMDL, is made, but no new parameter-table is required because the PARMS table already contains the information. The results are stored in row 2 of the PEANUTFIT table.

Finally, MKMODEL is used to simulate peanut oil concentrations with a 10-minute lag time between ingestion and the start of absorption. A new model-table, SIMMDL, is made. Notice that the DATA is specified as a LIST of X values from 0 to 120 minutes in steps of 5 minutes (LIST (0:120:5)). The model has been specified as a system of two equations. PLASMA is the main function, and it uses the subfunction, E, to describe the two exponential terms. Notice how the rate constants CLP/VD and KA are passed by changing the argument list sequence when E is called by PLASMA. Expressions such as CLP/VD and KA are quite acceptable as arguments (or even other functions). This model could be used for curve fitting the peanut oil data, and would let you estimate the lag time.

The following examples assume that you have already RAC'ed MKMODEL, called it, and have set the SIMULATION option to FALSE. This series of commands is shown in Figures 1-3.

Curve Fitting with a System of Differential Equations

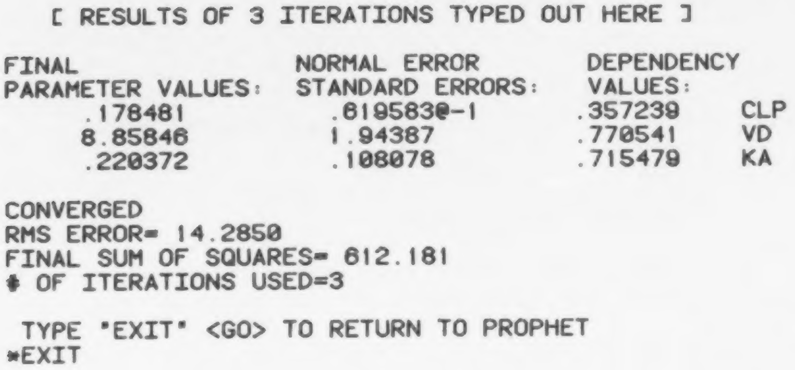
The only table MKMODEL needs before using a model-table is a data-table containing peanut oil concentrations at different times. In the examples below, this table is called PEANUT (see table CHRIS in the *PROPHET Primer*).

After making the model-table, DEQMDL, and the parameter-table, PARMS, the procedure recognizes that you want to perform curve fitting. It asks for further information about the parameters you wish to estimate and any constraints you wish to use.

Whether you use MKMODEL for simulation or curve fitting, the procedure then asks for the name of a table to contain the results. This may be a new name or the name of any existing PROPHET table. See Figures 7 and 8 for details.

```
ENTER NAMES OF PARAMETERS TO BE ESTIMATED
[HIT <GO> TO ESTIMATE ALL PARAMETERS]:CLP,VD,KA<

ENTER 1 OR MORE CONSTRAINTS
[HIT <GO> FOR NO CONSTRAINTS]:<

ENTER NAME FOR TABLE TO SAVE RESULTS:PEANUTFIT<

MLAB INSTRUCTIONS :

USE MLDATA;
DEQRELERR=.001;
FUNCTION PLASMA DIFF T (T)=(KA*GUT - CLP*PLASMA)/VD;
INITIAL PLASMA (0.)=0;
FUNCTION GUT DIFF T (T)=-KA*GUT;
INITIAL GUT (0.)=DOSE;
FIT(CLP,VD,KA),PLASMA TO DATA1;
!PLASMA DIFF T,GUT DIFF T;
!;
!20;
!.001;
SAVE CLP,VD,KA,STANDARDERRORS,COVP,BBNPARAMS IN MLPOUT;
DATA0=INTEGRATE(PLASMA DIFF T,GUT DIFF T,DATA1 COL 1);
DATA1 COL(NCOLS(DATA1)+1)=DATA0 COL 2;
SAVE DATA1,RSQUARE,RESIDSS IN MLDOUT;

 TYPE "DO MLABGO" <GO>


*DO MLABGO
USING:DATA1,WTS1
USING:DOSE,CLP,VD,KA

TYPE DERIVATIVE NAME(S) OF DEQ SYSTEM FOR MODEL

INITIAL PARAMETER VALUES:
                .150000
                8.00000
                .200000

BEGIN ITERATION 1
BEGIN ITERATION 1.1, SUM OF SQUARES= 921.879
BEGIN ITERATION 1.2, SUM OF SQUARES= 615.935
BEGIN ITERATION 1.3, SUM OF SQUARES= 615.899
BEGIN ITERATION 1.4, SUM OF SQUARES= 615.899
CURRENT PARAMETER VALUES:
             .172676
             8.88447
             .223684
```

Figure 7. MKMODEL performing curve fitting using a system of differential equations.



```
   [ RESULTS OF 3 ITERATIONS TYPED OUT HERE ]

FINAL                NORMAL ERROR        DEPENDENCY
PARAMETER VALUES:    STANDARD ERRORS:    VALUES:
     .178481            .619583@-1       .357239     CLP
    8.85846             1.94387          .770541     VD
     .220372            .108078          .715479     KA

CONVERGED
RMS ERROR= 14.2850
FINAL SUM OF SQUARES= 612.181
# OF ITERATIONS USED=3

 TYPE 'EXIT' <GO> TO RETURN TO PROPHET
*EXIT
```

Figure 8. Results of curve fitting using a system of differential equations.

MKMODEL displays the results of the fit after returning from MLAB, and it asks you whether you want to see graphs of the fit or more information about the statistical measures of the fit. The results-table created by MKMODEL in this example is shown in Figure 11, which follows the next example.

Curve Fitting with an Integrated Equation

Figure 9 illustrates how you use MKMODEL to create a model-table to fit the same data using an integrated equation. After you indicate that you are not using differential equations, you enter the model function in the same way as before. MKMODEL asks no questions about the X and Y bounds because they are not required for integrated equation models. You use the same parameter-table (PARMS) because the parameter names are identical for each type of model. In this example, the same initial estimates are also used, so that the results of the two approaches can be compared.

```
>CALL MKMODEL<

DO YOU HAVE A MODEL TABLE?NO<
DO YOU WANT TO USE DIFFERENTIAL EQUATIONS?NO<



PLEASE SUPPLY THE FOLLOWING FOR EACH FUNCTION:
        1) NAME - (OPTIONAL)
        2) ARGS - ARGUMENT LIST
        3) DEF  - FUNCTION DEFINITION
        4) DATA - DATA TABLE

         [TYPE 'EXIT' WHEN FINISHED]


FUNCTION (1):

NAME?PLASMA<
ARGS?T<
DEF?DOSE*KA/VD*(EXP(-CLP/VD*T)/(KA-CLP/VD)+EXP(-KA*T)/(CLP/VD-KA))<
DATA?PEANUT<
'T' COL NUMBER?1<
Y COL NUMBER?2<
WEIGHTING FUNCTION?E^<
SCALING FACTOR?E^<

FUNCTION (2):

NAME?EXIT<

WHAT NAME DO YOU WANT FOR THE MODEL FUNCTION TABLE?INTMDL<
[MAKING MODEL TABLE 'INTMDL'... DONE]
DO YOU WANT TO EDIT THE MODEL FUNCTION?NO<

DO YOU HAVE A PARAMETER TABLE?YES<
PARAMETER TABLE NAME ?PARMS<
```

INTMDL 1R X 10C

0 NAME	1 ARGUMENT	2 DEFINITION
1. PLASMA	T	DOSE*KA/VD*(EXP(-CLP/VD*T)/(KA-CLP/VD)+EXP(-KA*T)-

0 NAME	3 X BOUND	4 Y BOUND	5 DATA	6 X	7 Y	8 ERROR	9 WEIGHT	10 SCALE
1. PLASMA			PEANUT	1	2			

Figure 9. Creating a model-table for a single integrated equation.

After making the model-table INTMDL and asking you to specify the parameter-table PARMS, MKMODEL asks you to enter the names of the parameters you wish to estimate. Specify the parameters in the same way as before. The procedure then prompts you for any constraints you wish to use. Finally, before calling MLAB to perform the fit, MKMODEL asks for the name of your results-table. The name you supply can be the name of an existing table (PEANUTFIT in this example) or the name of a table you want to create. Figure 10 illustrates this part of your interaction with the procedure.

Figure 10 also illustrates how MKMODEL calls MLAB to perform curve fitting. Notice the difference in the MLAB instructions when you use an integrated model.

```
ENTER NAMES OF PARAMETERS TO BE ESTIMATED
[HIT <GO> TO ESTIMATE ALL PARAMETERS]:CLP,VD,KA<

ENTER 1 OR MORE CONSTRAINTS
[HIT <GO> FOR NO CONSTRAINTS]:<

ENTER NAME FOR TABLE TO SAVE RESULTS:PEANUTFIT<

MLAB INSTRUCTIONS :

USE MLDATA;
FUNCTION PLASMA(T)=DOSE*KA/VD*EXP(-CLP/VD*T)/(KA-CLP/VD)+EXP(-KA*T)/(C-
LP/VD-KA));
FIT(CLP,VD,KA),PLASMA TO DATA1;
!;
!20;
!.001;
SAVE CLP,VD,KA,STANDARDERRORS,COVP,BBNPARAMS IN MLPOUT;
DATA1=POINTS(PLASMA,DATA1);
SAVE DATA1,RSQUARE,RESIDSS IN MLDOUT;

 TYPE "DO MLABGO" <GO>

*DO MLABGO$
USING:DATA1
USING:DOSE,CLP,VD,KA

INITIAL PARAMETER VALUES:
                  .150000
                 8.00000
                  .200000

BEGIN ITERATION 1
BEGIN ITERATION 1.1, SUM OF SQUARES= 923.127
BEGIN ITERATION 1.2, SUM OF SQUARES= 615.382
BEGIN ITERATION 1.3, SUM OF SQUARES= 615.346
BEGIN ITERATION 1.4, SUM OF SQUARES= 615.346
CURRENT PARAMETER VALUES:
              .172550
             8.89922
              .224586

   [ RESULTS OF 3 ITERATIONS TYPED OUT HERE ]

FINAL                  NORMAL ERROR          DEPENDENCY
PARAMETER VALUES:  STANDARD ERRORS:   VALUES:
      .178793            .620164@-1         .354768     CLP
     8.86838            1.93438            .768676     VD
      .221084            .107929            .713989     KA

CONVERGED
RMS ERROR= 14.2826
FINAL SUM OF SQUARES= 611.979
# OF ITERATIONS USED=3

 TYPE "EXIT" <GO> TO RETURN TO PROPHET
*EXIT
```

Figure 10. MKMODEL performing curve fitting using a single integrated equation.

Figure 11 shows the results-table PEANUTFIT after MKMODEL has performed two fits on the same set of data, using first the model-table DEQMDL and then the model-table INTMDL. Before displaying the results-table, MKMODEL asks whether you want to see graphs of the fit or more information about the statistical measures of the fit.

```
WOULD YOU LIKE TO SEE GRAPHS OF THE FIT AND RESIDUALS?NO<
WOULD YOU LIKE TO SEE PARAMETER AND FIT STATISTICS?NO<
```

PEANUTFIT 2R X 13C

0 MODEL TABLE	1 CLP	2 VD	3 KA	4 MULT R**2	5 LOG L	6 SS RESIDUAL
1. DEQMDL	.1784814	8.858464	.2203716	.8646616	-22.38944	612.1808
2. INTMDL	.1787935	8.868379	.2210843	.8647062	-22.38845	611.9789

0 MODEL TABLE	7 # OBS	8 NOTES	9 PARAMETER SE ARRAY	10 COVARIANCE MATRIX
1. DEQMDL	6.	CONVERGED (3),-	[.06195828,1.	[[1.881223E-5,-.000
2. INTMDL	6.	CONVERGED (3),-	[.06201645,1.	[[1.885379E-5,-.000

0 MODEL TABLE	11 FUNCTION	12 DATA SET	13 DATA TABLE NAME
1. DEQMDL	[(.2203716*G	[[[1.,24.,22.09649,1.9035	[PEANUT]
2. INTMDL	[1000.*.2210	[[[1.,24.,22.13391,1.8660	[PEANUT]

Figure 11. Results-table PEANUTFIT after two curve-fitting sessions.

Simulation

To use MKMODEL for simulation, you must change two of the model options before calling MKMODEL. You must set SIMULATION to TRUE, and because MLAB provides no informative messages during simulation, you should set INTERACTIVE to FALSE so that your intervention is not required to start MLAB and to return to PROPHET. As Figure 12 illustrates, you can set these model options by calling the subprocedure SETOPTION. MKMODEL then prompts you for all of the information it needs to create a model-table for simulation.

```
>SETOPTION<
TYPE "EXIT" WHEN FINISHED
OPTION?SIMULATION<
VALUE ?TRUE<
OPTION?INTERACTIVE<

VALUE ?FALSE<
OPTION?EXIT<

>CALL MKMODEL<

DO YOU HAVE A MODEL TABLE?NO<
DO YOU WANT TO USE DIFFERENTIAL EQUATIONS?NO<

PLEASE SUPPLY THE FOLLOWING FOR EACH FUNCTION:
        1) NAME - (OPTIONAL)
        2) ARGS - ARGUMENT LIST
        3) DEF  - FUNCTION DEFINITION
        4) DATA - DATA LIST OR DATA TABLE

         [TYPE "EXIT" WHEN FINISHED]


FUNCTION (1):

NAME?PLASMA<
ARGS?T<
DEF?DOSE*KA*( E((T-LAG),CLP/VD,KA) + E((T-LAG),KA,CLP/VD))<
DATA?LIST(0:120:5)<

FUNCTION (2):

NAME?E<
ARGS?T,K1,K2<
DEF?EXP(-K1*T)/(VD*(K2-K1))<
DATA?E^<

FUNCTION (3):


NAME?EXIT<

WHAT NAME DO YOU WANT FOR THE MODEL FUNCTION TABLE?SIMMDL<
[MAKING MODEL TABLE "SIMMDL"... DONE]
DO YOU WANT TO EDIT THE MODEL FUNCTIONS?NO<
```

Figure 12. Changing model options and creating a model-table for simulation.

Note in Figure 12 that, after you enter the main function, PLASMA, you can type the MLAB LIST command in response to the prompt,

DATA?

You can then type the subfunction, E; because no data are required for a subfunction, you use CTRL-E to indicate an EMPTY value.

After making the model-table, SIMMDL, MKMODEL asks whether you have a parameter-table. Once again, you use the table PARMS. An additional row named LAG with the value of 10.0 has been added to the parameter table, PARMS, to specify a value for the lag parameter in this model (not shown). Finally, you specify that the results should be saved in a table called PEANUTSIM, and MKMODEL calls MLAB to perform the simulation. See Figure 13 for details.

```
DO YOU HAVE A PARAMETER TABLE?YES<
PARAMETER TABLE NAME ?PARMS<

ENTER NAME FOR TABLE TO SAVE RESULTS?PEANUTSIM<

MLAB INSTRUCTIONS :

DATA1=LIST(0:120:5);
FUNCTION PLASMA(T)=DOSE*KA*( E((T-LAG),CLP/VD,KA) + E((T-LAG),KA,CLP/V-
D));
FUNCTION E(T,K1,K2)=EXP(-K1*T)/(VD*(K2-K1));
DATA1=POINTS(PLASMA,DATA1);
SAVE DATA1 IN MLDOUT;
```

Figure 13. MKMODEL prompting you for parameter- and results-tables and then performing simulation.

The results of this simulation are shown in the graph in Figure 14.

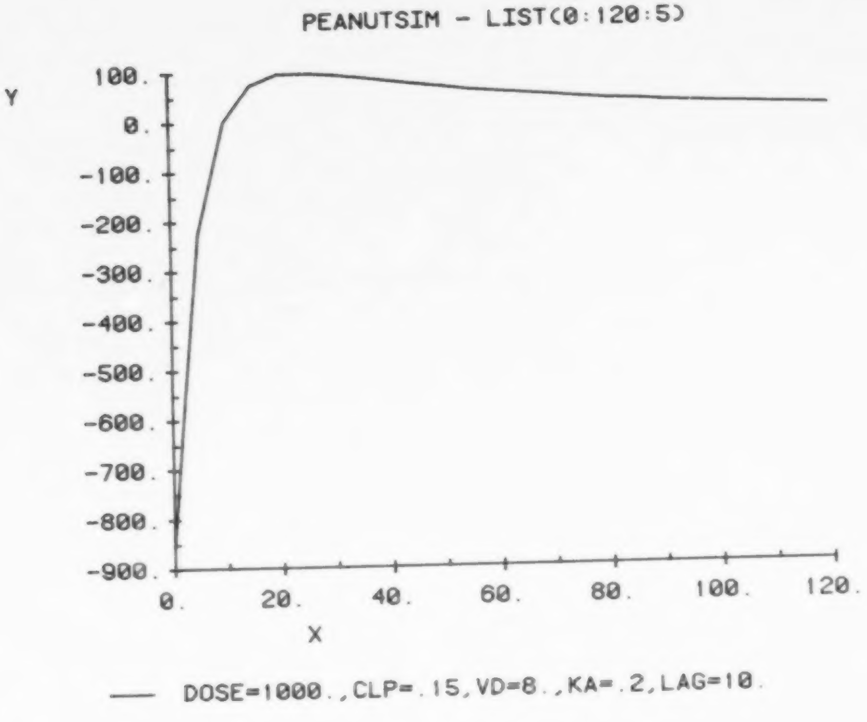


Figure 14. Graph of MKMODEL's simulation results.

In Figure 14, notice the negative concentrations predicted at times before 10 minutes (the lag time). Although mathematically correct, the model is clearly wrong for the description of drug concentrations at times before the lag time is over. To remedy this, change the model (using EDITMDL) to make the values before the lag equal to zero. Use the conditional function shown in the MLAB instructions above the graph in Figure 15. See how the revised model simulates expectations more reasonably.

```
MLAB INSTRUCTIONS :

DATA1=LIST(0:120:5);
FUNCTION PLASMA(T)=IF (T - LAG) < 0 THEN
   0
ELSE
   DOSE*KA*( E((T-LAG),CLP/VD,KA) + E((T-LAG),KA,CLP/VD));


FUNCTION E(T,K1,K2)=EXP(-K1*T)/(VD*(K2-K1));
DATA1=POINTS(PLASMA,DATA1);
SAVE DATA1 IN MLDOUT;
```

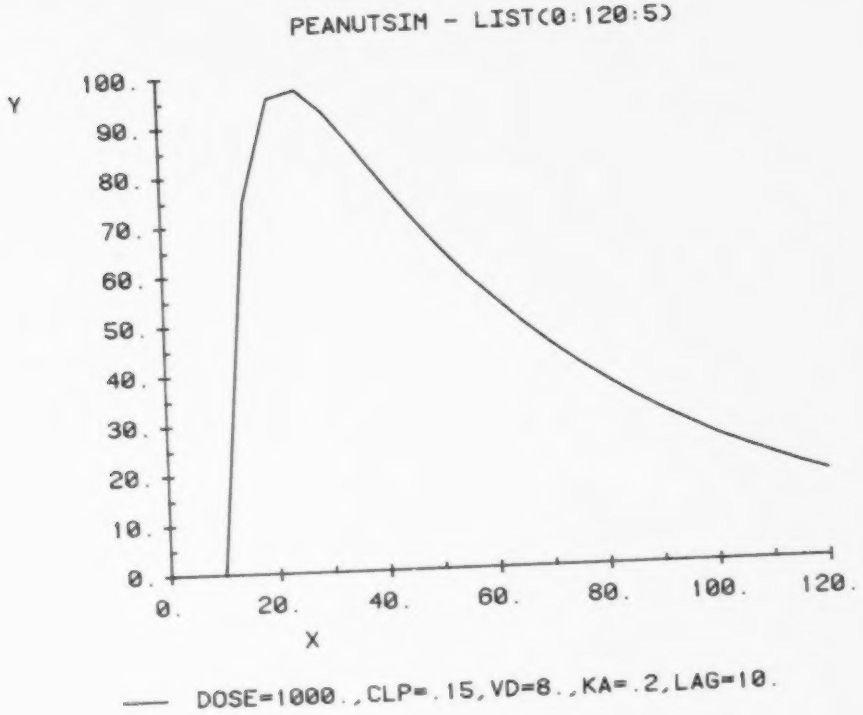


Figure 15. The revised simulation avoiding negative concentrations.

GOFIT

by Nick Holford
Division of Clinical Pharmacology
University of California at San Francisco

Summary

GOFIT displays goodness-of-fit measures for models fitted with the public procedures DRUGFUN and MKMODEL. Parameter estimates, standard errors, and the standard deviation of the fit are presented in a goodness-of-fit table.

Calling Sequence

```
RAC PUBLIC GOFIT <GO>
CALL GOFIT <GO>
```

Entry

```
CALL GOFITQT(results-table)
                <table>
```

Citation

Holford NHG. GOFIT, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., January 1982.

Questions or Comments

Send a message to 'HOLFORD' using PROPHET.

Things to Remember

GOFIT works only with a results-table produced by DRUGFUN or a results-table produced by MKMODEL.

Goodness-of-Fit Table

The public procedure GOFIT displays goodness-of-fit measures for models fitted with the public procedures DRUGFUN and MKMODEL.

When you call GOFIT, the procedure asks you for the name of the results-table produced by DRUGFUN or MKMODEL. It then displays a table containing the goodness-of-fit information. The table consists of one column for each parameter estimate and one column for each parameter's standard error (SE). The last column of the table contains the standard deviation (SD) of the fit. The goodness-of-fit table has rows corresponding in number and in name to those in the results-table; that is, it contains one row for each data set (see the example below).

The goodness-of-fit table is a temporary table. If you want to save it for future PROPHET sessions, you must answer YES to GOFIT's second question:

```
DO YOU WANT A TOP-LEVEL VERSION OF THIS
 TEMPORARY TABLE?
```

If you do, GOFIT asks you to supply a name for it.

Figure 1 illustrates how you would use GOFIT to obtain goodness-of-fit measures for the results-table called PEANUTFIT. (PEANUTFIT comes from the MKMODEL example in this volume.)

```
>RAC PUBLIC GOFIT<

>CALL GOFIT<

FIT TABLE NAME?PEANUTFIT<



 GOODNESS OF FIT - PEANUTFIT  2R X 7C
+-----------+----------+-----------+----------+----------+----------+----------+
|0          |1         |2          |3         |4         |5         |6         |
|FUNCTION   |CLP       |CLP        |VD        |VD        |KA        |KA        |
|TABLE      |          |SE         |          |SE        |          |SE        |
+-----------+----------+-----------+----------+----------+----------+----------+
|1. DEQMDL  |.1784814  |.06195828  |8.858464  |1.943874  |.2203716  |.1080781  |
|2. INTMDL  |.1787935  |.06201645  |8.868379  |1.934376  |.2210843  |.1079291  |
+-----------+----------+-----------+----------+----------+----------+----------+
+-----------+----------+
|0          |7         |
|FUNCTION   |FIT       |
|TABLE      |SD        |
+-----------+----------+
|1. DEQMDL  |14.28497  |
|2. INTMDL  |14.28261  |
+-----------+----------+


 DO YOU WANT A TOP-LEVEL VERSION OF THIS TEMPORARY TABLE?NO<
```

Figure 1. Using GOFIT.

Quiet Entry GOFITQT

GOFIT has a quiet entry, GOFITQT, which takes the name of your results-table as its argument. You should use the quiet entry in conjunction with other commands (such as DISPLAY) or within another procedure. GOFITQT neither displays the goodness-of-fit table nor produces a top-level copy of it.

Interpreting Goodness-of-Fit

You should be careful when you interpret the statistical measures of the fits of nonlinear models. In particular, if you have specified constraints and a parameter estimate lies at or near a constraint boundry, the standard error of the parameter estimate is invalid in the usual sense.

For example, if you use the constraint $A \geq 0$ for the parameter A and the estimated value of A is 0, then A lies at the constraint boundary and its standard error estimate is invalid for statistical testing or estimation of confidence limits. Similarly, if you have actively constrained any parameter value, the sum of squares minimization is altered and the standard deviation of the fit is no longer a reasonable estimate.

In general, you should avoid statistical hypothesis testing when you have fit nonlinear models. The standard errors are useful, however, as semiquantitative indicators of how well a particular parameter has been estimated. Standard errors that are large in relation to the parameter value may be due to:

- errors in the data supplied for the fit
- the wrong model being fitted
- a model too complex for the available data (overspecified model)
- high correlation between parameters

You can use the standard deviation of the fit in conjunction with the coefficient of determination (multiple R-squared) and the residual sum of squares to evaluate how well the model fits the data overall.

GRAFIT

by Nick Holford
Division of Clinical Pharmacology
University of California at San Francisco

Summary

GRAFIT graphically displays the data in results-tables made by the public procedures DRUGFUN and MKMODEL. It makes graphs showing the observed and fitted values, and it has an option to display plots of the residuals against the dependent and independent variables. If you want to make an individual graph of data contained in a results-table, you can call the procedure GETMGRAPH, which is used by GRAFIT.

Calling Sequence

```
RAC PUBLIC GRAFIT <GO>
CALL GRAFIT <GO>
```

Entries

```
CALL QG <GO>

                                                            Xnumber,
CALL GRAFITQT(results-table, dismode,      rows,            <fixed>
                 <table>     <fixed>   <fixed array>          or
                                                          <fixed array>

          low,      high,    logaxis)
        <float>   <float>    <text>
```

Citation

Holford NHG. GRAFIT, in <u>PROPHET Public Procedures Notebook</u> (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Send a message to 'HOLFORD' using PROPHET.

Things to Remember

GRAFIT shows the simulated or fitted values connected with straight lines. If you have relatively few data points, the appearance of the curve may differ from the shape of a curve drawn through all points predicted in the range of the independent variable.

Evaluation of a fitted model -- like one of those produced by DRUGFUN -- usually requires inspection of a graph of the observed and fitted values. You can best assess the adequacy of the model by examining such a graph in conjunction with plots of the residuals against the dependent and independent variables. The public procedure GRAFIT generates these graphs for data in DRUGFUN results-tables.

You can also call GRAFIT to plot simulations of functions produced by MKMODEL. When used for this purpose, GRAFIT requires all the same arguments that it needs to generate graphs of a fitted model. Of course, the procedure produces no residual plots for MKMODEL function simulations.

GRAFIT Arguments

GRAFIT's seven arguments are described below. The interactive version prompts you for each argument. The quiet entry, GRAFITQT, requires that you supply all seven arguments when you call the procedure.

Results-table

GRAFIT works only with a results-table made by MKMODEL or DRUGFUN, or with a table conforming to the format of these results-tables. The data tables used for the model need not be available, because the results-table contains a copy of the data.

GRAFIT uses several parts of results-table to construct a title for each set of graphs it produces. It takes the name of the results-table itself, the name of the row in results-table containing the data GRAFIT displays, and the first two footnotes of results-table, if they exist.

If results-table has footnotes, GRAFIT includes the first two of them in the title it gives to the graphs it produces. DRUGFUN automatically provides the name of the DRUGFUN model in footnote 1 and the weighting function in footnote 2. You may want to change the footnotes to provide a more appropriate title.

Dismode

The dismode (display mode) options are:

1 = DISPLAY
2 = PRINT
3 = PLOT

If you specify dismode 1, GRAFIT simply displays the graphs it produces. If you choose dismode 2, the procedure displays and prints the graphs. If you specify dismode 3, the procedure displays the graphs and creates plotted copies, which are sent to you by mail.

Rows

Results-table may contain more than one row. GRAFIT asks whether you want to graph the data in all of them. If you answer NO, you must specify the desired rows by number.

When you use the quiet version of GRAFIT, you can select one or more rows for display by giving the row numbers in a FIXED ARRAY as the third argument of GRAFITQT. If you leave this argument empty, the procedure displays the curves in all of the rows.

Xnumber

If DRUGFUN or MKMODEL used more than one independent (X) variable in the fit, then GRAFIT needs to know which X variable to graph against the dependent (Y) variables. Each element in Xnumber, the X-variable index, is a sequence number in the list of X variables specified for the model.

If Xnumber is a single FIXED number, GRAFIT uses the same X variable for all the rows of the results-table that you specified with the rows argument. If you want to use different independent variables for different rows, Xnumber should be a FIXED ARRAY containing the X-variable numbers corresponding to the row numbers in the rows argument. If you do not specify Xnumber, GRAFIT uses the first independent variable for all rows.

Low

You may want to examine a particular range of X values in the graph of the display. The interactive version of GRAFIT gives you this option by asking:

```
WOULD YOU LIKE TO SEE A WINDOW OF THIS GRAPH
 AND RESIDUAL PLOTS?
```

If you answer YES, GRAFIT further prompts you to

```
ENTER LOW AND HIGH LIMITS FOR X-AXIS WINDOW
```

The procedure then displays a graph showing only the selected range alongside the graph of the whole curve.

The quiet entry GRAFITQT automatically displays a window of the graph and residual plots. If you do not specify the low argument, the default is the minimum X value in the original data.

High

Specify the upper limit of the X range to be shown on the additional graph with the FLOAT argument, high. If you leave high empty, its default value is 25% of the maximum X value in the original data.

Logaxis

You can use logarithmic axes to display the X variables, the Y variables, or both. Simply specify X, Y, or XY as the value for the TEXT argument logaxis. GRAFITQT uses Cartesian axes if logaxis is empty.

Your specification of logaxis does not affect graphs of residuals. Remember that if the fit used unequal weighting, the residuals are weighted residuals!

Quick Interactive Entry, QG

GRAFIT has a quicker interactive version, QG, that lets you see the fit(s) without displaying residuals and without giving you the option of seeing graph windows.

QG asks you for the values of only four arguments: results-table, dismode, logaxis, and rows (if results-table has more than one row). If results-table has two or more rows and you give no row numbers, QG displays only the data in the last row. If you reply ALL to the row number prompt, all the rows are graphed.

GETMGRAPH

If you want to make an individual graph from the data contained in a results-table, use the procedure called GETMGRAPH. GRAFIT uses this procedure, and it is RAC'ed automatically whenever you call GRAFIT or QG. You can obtain it directly by issuing these commands:

```
ACCESS PUBLIC MODELSAVED  <GO>
RAC GETMGRAPH FROM MODELSAVED <GO>
CALL GETMGRAPH <GO>
```

When you call GETMGRAPH, the procedure asks you to supply the name of a model results-table and to specify the kind of graph you want to make. You must also give your graph a name.

There are five kinds of graphs:

- P - a graph of predicted values obtained from simulation or curve fitting
- O - a graph of observed values used for curve fitting
- F - a graph combining O and P curves to compare the fit of predicted to observed values
- X - a residual plot against the X-variable values
- Y - a residual plot against the observed Y-variable values

GETMGRAPH can make only a P graph from the results of a simulation. The default value is F for the results of curve fitting.

If the results-table has more than one row, GETMGRAPH prompts you for the row you want to use. The default value is the last row in the table.

If there is more than one data set in each row, the procedure prompts you for the number of the data set in the row. This number corresponds to the model-table row used to generate the results. The default value is the first nonempty data set.

If there is more than one X variable in the data, GETMGRAPH asks you for the number of the X variable. The default value is 1.

The procedure then makes a graph and asks you to give it a top-level name. The graphs made by GETMGRAPH are similar to those produced by GRAFIT except for the logaxis option.

Quiet GETMGRAPH Entry

You can use GETMGRAPH quietly by giving the command:

```
G = GETMGRAPHQT(resultab, grafspec, rowdataX)
```

where G is the name of the graph the procedure returns, and the three arguments are as defined in Table 1.

Table 1. GETMGRAPHQT Arguments

Argument	Type	Value	Default value
resultab	<table>	results-table name	(none)
grafspec	<text>	'P','O','F','X', or 'Y' if model is a curve fitting;	'F'
		'P' if model is a simulation	'P'
rowdataX	<fixed array>	[row number, data set number, X-variable number]	[last row, first non-empty set, 1]

For example,

```
G = GETMGRAPHQT(MYFIT, 'X', [1, 2, 3]) <GO>
```

returns a graph of residuals plotted against the third X variable in data set number 2 in row 1 of a results-table called MYFIT.

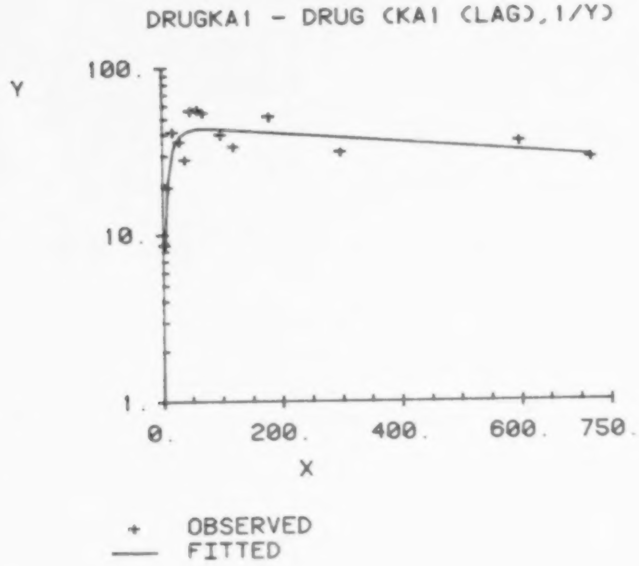
GRAFIT Example

The example below illustrates the use of GRAFIT to examine the results of nonlinear curve fitting using DRUGFUN. You can use GRAFIT in the same way to examine the results of MKMODEL.

DRUGFUN fits the concentration-time data contained in a table called DRUG to a first-order absorption, one-compartment model (KA1). A results-table called DRUGKA1 contains the results of the fit. (See the DRUGFUN example in this notebook for details.)

GRAFIT displays a graph of the fit after you choose a logarithmic scale for the Y axis. The procedure takes the title of the graph from the name and footnotes of the results-table, DRUGKA1. DRUGFUN fits these data to a KA1 model with an estimate of the lag time and a weighting function of 1/Y. A straight line connects the fitted values, and the corresponding observations are displayed as crosses. If you want a top-level copy of the graph to examine or edit, you must give the graph a name at this point.

```
>RAC PUBLIC GRAFIT<
>CALL GRAFIT<

WHAT IS THE NAME OF THE RESULTS TABLE?DRUGKA1<

DISPLAY MODE:
         1 = DISPLAY
         2 = PRINT
         3 = PLOT?1<

DO YOU WANT TO USE LOG SCALES FOR AXES?Y<
X-AXIS ONLY (TYPE "X")?
Y-AXIS ONLY (TYPE "Y")?
BOTH AXES (TYPE "XY")?Y<
```



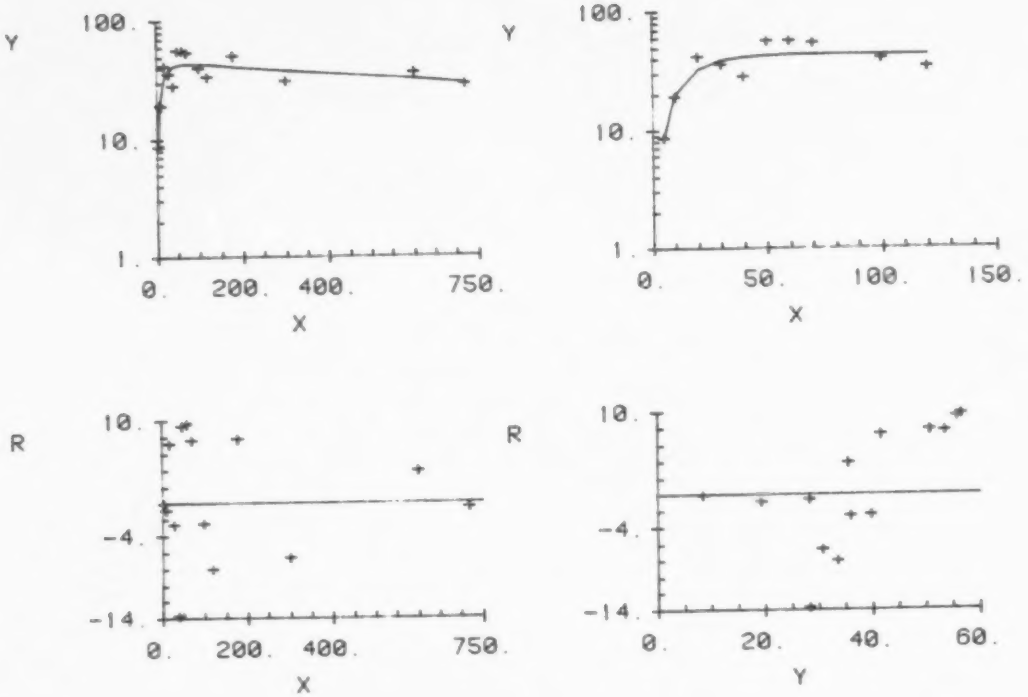
```
DO YOU WANT A TOP-LEVEL VERSION OF THIS GRAPH?NO<
WOULD YOU LIKE TO SEE A WINDOW OF THIS GRAPH AND RESIDUAL PLOTS?YES<
```

Figure 1. Using GRAFIT to produce a graph of fitted data.

The final display includes four graphs: a scaled-down version of the original graph; a window of the fit made by specifying X-axis limits of 0. and 150.; and two plots of the (weighted) residuals against the independent (X) and dependent (Y) variables. Once again, the name of data-table and details of the model and weighting function are shown in the title.

```
ENTER LOW AND HIGH LIMITS FOR X-AXIS WINDOW
LOWER LIMIT?0<
UPPER LIMIT?150<
```

```
DRUGKA1 - DRUG (KA1 (LAG),1/Y)
```



```
TYPE "PLOT" FOR A PLOT OR HIT <GO> TO PROCEED?<
```

Figure 2. Graphs produced by GRAFIT: original fit, fit window, and two residual plots.

MODELTEST

by Nick Holford
Division of Clinical Pharmacology
University of California, San Francisco

Summary

MODELTEST compares a series of two or more models that have been used to fit curves to a common set of observations. You can use either the Schwartz criterion or the F-test to decide which is the optimal model.

Calling Sequence

```
RAC PUBLIC MODELTEST <GO>
CALL MODELTEST <GO>
```

Citation

Holford NHG. MODELTEST, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Send a message to 'HOLFORD' using PROPHET.

Things to Remember

MODELTEST uses results-tables from MKMODEL. It also works with DRUGFUN results-tables, but should only be used to compare models fitted with the same weighting scheme.

Estimation of Log Likelihood

The likelihood, L, is defined by

$$L = \prod_{i=1}^{N} \left[\frac{W_i}{2\pi\sigma^2} \right]^{1/2} \cdot e^{\frac{-(Y_i - \hat{Y}_i)^2 \cdot W_i}{2\sigma^2}}$$

where N = number of observations
σ^2 = variance
Y_i = observed value for the ith observation
$\hat{Y}_i$ = predicted value for the ith observation
W_i = weight value for the ith observation

An estimate of σ^2 is given by:

$$\hat{\sigma}^2 = \frac{1}{N} \sum_{i=1}^{N} (Y_i - \hat{Y}_i)^2 \cdot W_i$$

The log likelihood is obtained from the definition of likelihood:

$$\log L = \frac{1}{2} \sum_{i=1}^{N} \log W_i - \frac{N}{2} \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (Y_i - \hat{Y}_i)^2 \cdot W_i$$

By defining the weighted sum of squared residuals to be

$$SSQ = \sum_{i=1}^{N} (Y_i - \hat{Y}_i)^2 \cdot W_i$$

and the sum of the log of the weights to be

$$LOGWTSUM = \sum_{i=1}^{N} \log W_i$$

then

$$\hat{\sigma}^2 = \frac{SSQ}{N}$$

and

$$\log L = \frac{LOGWTSUM}{2} - \frac{N}{2} \log 2\pi\sigma^2 - \frac{SSQ}{2\sigma^2}$$

Substituting for σ^2 with its estimate $\hat{\sigma}^2$ yields:

$$\log L = \frac{1}{2} \left[LOGWTSUM - N \log \frac{2\pi SSQ}{N} - N \right]$$

Standardization of Weights

If you choose the values of the weights so that their extended product equals 1, then the value of LOGWTSUM becomes zero. You can then compute log L directly from SSQ using these "standardized" weights and N.

You can compute the standardized weight, W_s, from an arbitrary weight, W, with the expression:

$$W_s = W \cdot e^{-LOGWTSUM/N}$$

Notice that the standardized weight is 1 if all the arbitrary weights are equal to 1.

Selection of the Best Model

A variety of methods have been proposed to help in the selection of the best of a series of contending models. These models include the F-test (Boxenbaum et al., 1974) and the AIC (Akaike, 1979). The F-test should be applied only to models that are progressive simplifications of a generalized model and that, ideally, are linear in all of the parameters. The AIC overcomes the limitation of linearity, but is restricted to comparing hierarchical models.

You can use a newer test, the Schwartz criterion, to test a series of arbitrarily defined models that may be nonlinear in their parameters (Schwartz, 1978). This test assumes that the number of observations is sufficiently large to allow an asymptotic approximation. Leonard (1979) has proposed a method for determining whether this assumption is reasonable. If Leonard's test fails for a particular series of models, then you should conclude that the Schwartz criterion is inappropriate for comparing them. At present, there is no general alternate approach to help you in this situation, but the failure of Leonard's test may mean that you are asking too much of your observations with the particular models you have tried.

The Schwartz criterion, SC, is computed from:

$$SC = \log L - \frac{K}{2} \ln N$$

where log L is the log likelihood, K is the number of parameters that have been estimated in the model, and N is the number of observations.

Leonard's test is applied by computing the Leonard criterion, LC:

$$LC = SC + \frac{1}{2}\left[K \ln(2\pi N) + \ln(DCM)\right]$$

where DCM is the determinant of the covariance matrix. If the largest value of SC and the largest value of LC, computed for a series of models, come from the same model, then it is appropriate to use the Schwartz criterion.

The F-test assumes that the errors are normally distributed and is strictly applicable only to a series of models that form some kind of hierarchy, e.g., a series of polynomial models of increasing order. You should not use this test to compare models that differ in nonlinear parameters, e.g., one-compartment versus two-compartment pharmacokinetic models that are nonlinear in the exponential rate constant. The F statistic is computed as follows (Neter and Wasserman, 1974):

$$F = \frac{(SSR - SSG) \cdot (N - GP)}{SSG \cdot (GP - RP)}$$

The two models being compared are known as the general model (more parameters) and the reduced model (submodel of the general model, with fewer parameters). The residual sum of squares is SSG for the general model, and SSR for the reduced models. The number of observations is N for both models. The number of parameters is GP in the general model, and RP in the reduced model.

Using MODELTEST to Select the Best Model

When you call MODELTEST, the procedure gives you the choice of two methods for selecting the best of a series of models. The Schwartz criterion is the most generally applicable, but may fail to give you an answer; you can use the F-test if the models satisfy the assumptions discussed above. You can apply the Schwartz criterion to a series of two or more models that have been fitted to the same set of observations using MKMODEL or DRUGFUN. You can use the F-test to compare only two models at a time, one of which (reduced) is a mathematical simplification of the other (general).

Schwartz Criterion

If you select the Schwartz criterion, MODELTEST prompts you for the names of the model results-tables. Each results-table may have one or more rows, but each row in different tables must contain the results of the fit to the same set of observations. For example, if you have fitted two models, A and B, to each of six sets of data and the results of fitting model A to each set are contained in a table called ARESULTS and the results of fitting model B to each set are contained in BRESULTS, then the fits to data set number 1 must be in row 1 of ARESULTS and row 1 of BRESULTS, and so on for each data set up to and including row 6.

MODELTEST displays the results of using the Schwartz criterion in a temporary table. Column 1 contains the name of the results-table with the best model for each row of the series of results-tables. If the Leonard test fails to confirm the appropriateness of using the Schwartz criterion, the entry in the table is empty.

The remaining columns in the temporary table contain two values in each column. The first is a standardized Schwartz criterion obtained by dividing each of the entries in the table by the absolute mean Schwartz criterion. The use of a standardized value does not alter the meaning of the criterion, because the criterion is based only on the rank of the value in a series. Standardization allows easier comparison of the relative ranks.

The second value in each column is the difference between the Leonard and Schwartz criteria, also standardized by division by the absolute mean Schwartz value. The larger this value, the more emphasis is given to selecting a model on the basis of more precise estimates of the parameters, unlike the Schwartz criterion, which is based only on the residual sum of squares and the number of parameters. The largest value is associated with the best model.

After scanning this table, MODELTEST gives you the option of creating a top-level copy of it. Remember, temporary tables disappear when you log out. See Figure 1 for details.

```
>RAC PUBLIC MODELTEST<

>CALL MODELTEST<

MODELTEST - WHICH METHOD DO YOU WANT TO USE:-

        (1) SCHWARTZ-LEONARD CRITERIA
        (2) F-TEST

  (TYPE "1" OR "2")?1<

ENTER RESULT TABLE NAMES - TYPE "EXIT" WHEN FINISHED

        MODEL #1?PEANUTFIT<
        MODEL #2?PEANUTFIT2<
        MODEL #3?EXIT<



DO YOU WANT A TOP-LEVEL VERSION OF THIS TEMPORARY TABLE?N<




            [SCHWARTZ,LEONARD] CRITERIA - AVERAGE = -23.31619
            (K=NUMBER OF PARAMETERS IN MODEL)  2R X 3C
            +----------++-----------+---------------+---------------+
            |0         ||1          |2              |3              |
            |SOURCE    ||OPTIMUM    |PEANUTFIT      |PEANUTFIT2     |
            |TABLE     ||  MODEL    |K=3            |K=3            |
            +==========++===========+===============+===============+
            |1. DEQMDL ||PEANUTFIT2 |[-1.08,-.331]  |[-.925,-.196]  |
            +----------++-----------+---------------+---------------+
            |2. INTMDL ||PEANUTFIT2 |[-1.08,-.331]  |[-.924,-.199]  |
            +----------++-----------+---------------+---------------+
```

Figure 1. Using the Schwartz-Leonard criteria to compare models.

F-test

If you choose to use the F-test, MODELTEST asks you to supply the names of the results-tables for the general and reduced models. The procedure displays the F-test results in a three-column temporary table. Column 1 contains the names of the models, column 2 holds the value of F for each model, and column 3 contains a probability statement for accepting the null hypothesis of no difference between the two.

You can save a top-level version of this temporary table.

References

Boxenbaum HG, Riegelman S, and Elashoff RM. Statistical estimations in pharmacokinetics. J Pharmacokinet Biopharm 2:123-148, 1974.

Akaike M. A Bayesian extension of the minimum AIC procedure of auto regressive model fitting. Biometriika 66:237-242, 1979.

Schwartz G. Estimating the dimensions of a model. Ann Stat 6:461-464, 1978.

Leonard T. Why do we need significance levels? MRC Technical Report, University of Wisconsin, 1979.

Netter J, Wasserman W. Applied Linear Statistical Models. Irwin, Homewood, IL, 1974, p. 264.

best copy available

CELLSIM

by
Charles Donaghey
University of Houston
and
Mark Lavin and Jon Sutton
Bolt Beranek and Newman Inc.

Summary

CELLSIM simulates patterns of growth in cell populations. You can define the population to be simulated using a free-form text. The results of the simulation are presented in PROPHET tables and graphs.

Calling Sequence

```
RAC PUBLIC CELLSIM <GO>
CALL CELLSIM <GO>
```

Entry

```
CALL RBCELLSIM
```

Citation

Donaghey C, Lavin M, and Sutton J. CELLSIM, in PROPHET Public Procedures Notebook (HM Perry, ed.) Bolt Beranek and Newman Inc., Cambridge, MA, January 1982.

Questions or Comments

Contact Jon Sutton on the PROPHET line.

Things to Remember

This documentation assumes that you are already familiar with the CELLSIM simulation language as described in Donaghey's CELLSIM II User's Manual (see reference at the end of the CELLSIM documentation). Many of the results produced by CELLSIM are displayed in the form of PROPHET graphs, which look considerably better when rendered on a display terminal.

CELLSIM provides you with a way to simulate patterns of growth and change in populations of cells. CELLSIM was originally developed as a FORTRAN program by Charles Donaghey. The public procedure CELLSIM described here is actually a PL/PROPHET interface procedure; it sets up input files for the FORTRAN program, invokes that program, and then converts the simulation results, stored in disk files, into PROPHET tables and graphs.

Donaghey has written an extensive manual that describes the CELLSIM input language. You can use that language, which consists of free-format commands, to do all of the following:

- define the various cell populations
- define the growth states within those populations
- specify the parameters that determine transitions between various states and populations
- request that various aspects of the simulation be made available as program output

This documentation concentrates on using CELLSIM in the PROPHET environment; it assumes that you are already familiar with the CELLSIM simulation language, as described in Donaghey's CELLSIM II User's Manual.

Supplying a Source Text

When you call CELLSIM, the procedure asks you the following question:

```
WHERE IS THE TEXT OF THE CELLSIM SOURCE PROGRAM?

      1.  SAVED IN A PRIVATE SAVED TABLE,

      2.  SAVED IN A NAMED SAVED TABLE,

      3.  MAKE A NEW TEXT,

      4.  EXIT FROM CELLSIM

      (TYPE 1, 2, 3, OR 4)
```

The TEXT OF THE CELLSIM SOURCE PROGRAM refers to a PROPHET text that contains the list of CELLSIM commands (see Figure 1).

```
CELL TYPES TEST;
STATES(1)G1,S,G2,M;
FLOW(1)G1-S(ALL),S-G2(ALL),G2-M(ALL),M-G1(ALL);
TIMEINSTATES(1)G1:NORMAL(7,2),S:NORMAL(12,4),G2:NORMAL(5,2),M:NORMAL(-
1,.3);
PROLIFERATION(1)M:1.8;
INOCULUM(1)1000;
KILL(1)ATTIME=30 S:.95;
REPORTS 1 HOURS;
POSITION:10,40:S(1);
GRAPHTOTAL,S(1)/TOTAL,M(1);
HISTOGRAM:10,20,40:G1(1),G2(1),M(1);
FMF:11,21,41:G1(1),M(1)+G2(1),S(1):LABEL=
TEST-LABEL-FOR-FMF:CV=7,4,5;
SIMULATE 50 HOURS;
/
```

Figure 1. A CELLSIM source program.

If you choose Option 1, CELLSIM prompts you for the name under which the text is saved in your PRIVATE SAVED table.

If you specify Option 2, the procedure asks you for the name of the saved table and then prompts you for the name of the saved text, as in Option 1.

If you choose Option 3, CELLSIM places you in the PROPHET text editor so that you can create a new source text.

Option 4 allows you to exit if you decide not to do any simulation.

When you decide to use an existing source text (Options 1 and 2), CELLSIM asks whether you want to edit the text. If you answer YES, the procedure places you in the PROPHET text editor with a copy of the previously saved source text to modify. When you finish editing (creating or updating the source text), you are given the option of saving the source text. If you choose to save the text, CELLSIM prompts you for the name under which to save it.

Running the FORTRAN Simulation

At this point, the source text is ready to be used, and CELLSIM asks you:

DO YOU WANT TO RUN CELLSIM AS A BATCH JOB?

Running CELLSIM as a batch job defers execution of the simulation until a later time, when it will not interfere with the current terminal session, and when CPU-time rates are significantly lower.

If you answer NO, the procedure types:

```
[INVOKING CELLSIM FORTRAN PROGRAM...
```

and PROPHET transfers control to Donaghey's FORTRAN program to perform the simulation. When the simulation is completed, PROPHET erases the screen and types:

```
...RETURNING FROM FORTRAN]
```

Note: In the original output format, the FORTRAN program may abort without returning you to PROPHET. After some long messages, you will see a period (.) in the left margin as a prompt character. If this happens, give the command:

```
.CALL RUNRTN <GO>
```

This will return you to PROPHET.

When control returns to PROPHET, CELLSIM types out a list of errors detected by the FORTRAN program. For each error, the last input command in the source text scanned before the error is displayed. This may help you debug the source text.

Viewing the Simulation Results

Next, if you specified a REPORTS command in the source text, CELLSIM asks you:

```
DO YOU WISH TO SEE REPORTS PRODUCED
  BY CELLSIM?
```

If you answer YES, CELLSIM displays the REPORTS results in a PROPHET table, where each row corresponds to a time in the simulation, and the columns contain the numbers of cells in the various states and populations (see Figure 2). After scanning the REPORTS table, CELLSIM asks whether you want to make a permanent, top-level copy of the table. If you answer YES, the procedure prompts you for a name for the table.

CELLREPORT 51R X 7C

	1 SAMPLE (TIME)	2 TOTAL CELLS OVERALL	3 TOTAL CELLS IN TEST	4 CELLS IN G1 (TEST)	5 CELLS IN S (TEST)	6 CELLS IN G2 (TEST)	7 CELLS IN M (TEST)
1	0	1000	1000	267	480	214	39
2	1	1023	1023	307	468	214	33
3	2	1050	1050	331	504	155	59
4	3	1098	1098	414	492	156	36
5	4	1126	1126	442	504	168	12
6	5	1136	1136	412	542	134	47
7	6	1193	1193	495	539	123	36
8	7	1212	1212	490	550	148	24
9	8	1221	1221	444	568	160	49
10	9	1261	1261	419	657	160	25
11	10	1281	1281	408	688	148	37
12	11			431	625	243	12
13					602	280	12
14		1491				280	25
15	39	1494	1494				
	40	1499	1499				
42	41	1503	1503	330			
43	42	1503	1503	213	114		
44	43	1503	1503	52	1244	206	3
45	44	1506	1506	32	1244	229	1
46	45	1506	1506	30	1129	343	3
47	46	1510	1510	26	1102	358	24
48	47	1529	1529	63	1006	436	23
49	48	1549	1549	108	886	492	63
50	49	1635	1635	299	826	422	89
51	50	1705	1705	455	743	332	175

Figure 2. A CELLSIM reports table generated by the source program shown in Figure 1.

If you requested output using the GRAPHS command in the source text, CELLSIM next prompts you with the question:

```
DO YOU WISH TO SEE GRAPHS PRODUCED BY CELLSIM?
```

If you answer YES, the graphs are displayed. For each graph, CELLSIM gives you the option of making a top-level copy, which you can edit, display, or save. Figure 3 shows an example of a GRAPH produced by CELLSIM.

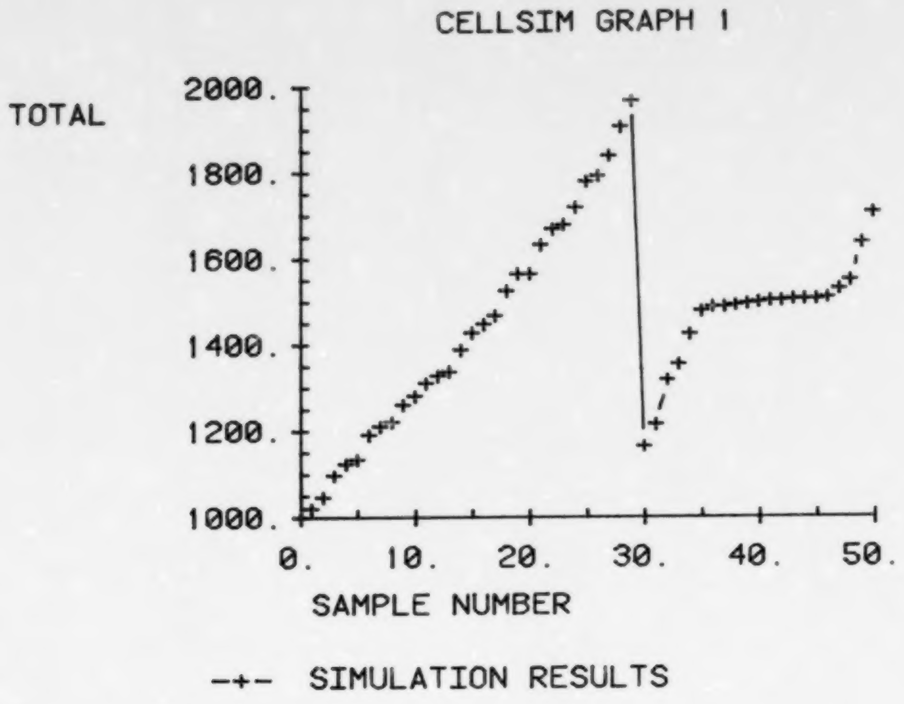


Figure 3. A CELLSIM graph generated by the source program shown in Figure 1.

If you requested output using the HISTOGRAM command, CELLSIM gives you the option of displaying and making top-level copies of the histogram graphs (see Figure 4).

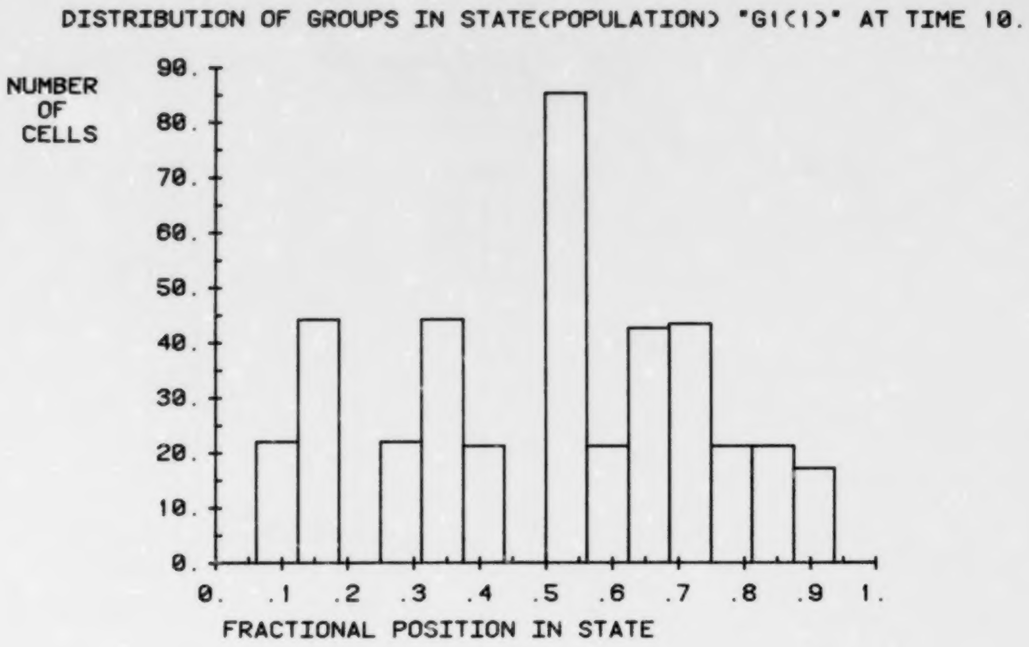


Figure 4. A CELLSIM histogram generated by the source program shown in Figure 1.

Note: In the original output format for the FORTRAN version of CELLSIM, all histograms at a particular sample time were displayed on the same X-axis. For simplicity, the PROPHET versions of the histograms are further subdivided, so that only one state's histogram appears on each graph.

If you requested simulated flow microfluorimeter output by using an FMF command in your source text, CELLSIM next asks you whether you want to see the FMF graphs.

If you answer YES, the requested FMF histograms are displayed. Again, you have the option of making a top-level copy of each (see Figure 5).

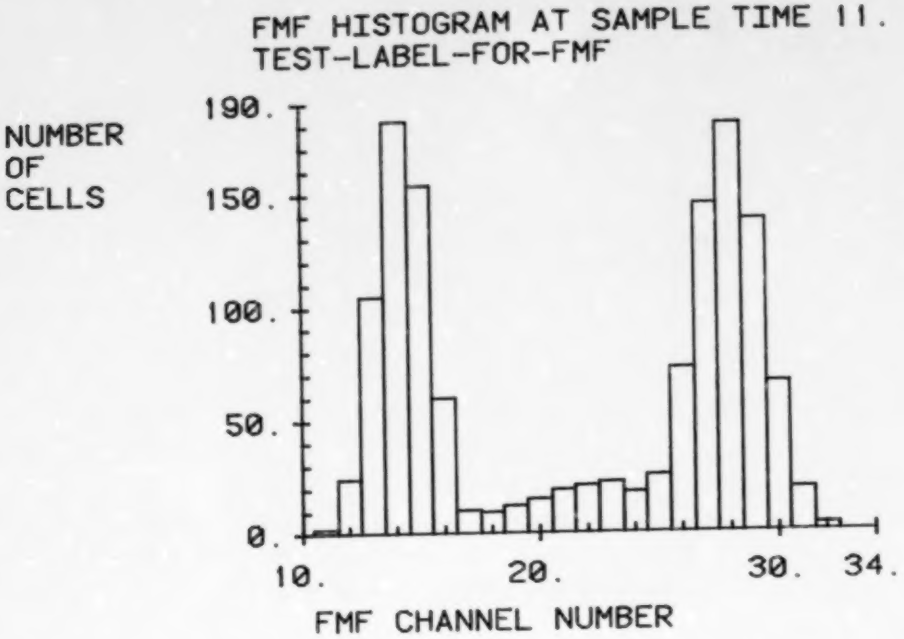


Figure 5. A CELLSIM FMF histogram generated by the source program shown in Figure 1.

Note that if you do make top-level copies, you can subsequently display the POINTS tables of the graphs to get the exact cell count in each channel.

If you requested output using a POSITION command, CELLSIM asks:

```
     DO YOU WISH TO SEE THE RESULTS OF
       THE "POSITION" COMMAND?
```

If you answer YES, CELLSIM scans the table containing the POSITION results (as shown in Figure 6).

GROUP POSITIONS AT 10. FOR S (1) 49R X 4C (NOTES)

	1 NUMBER CELLS IN GROUP	2 TIME OF ENTRY INTO STATE	3 TIME OF EXIT FROM STATE	4 FRACTION OF STATE COMPLETED
1	12.31891	-2.662474	12.49264	.8355251
2	12.31891	-2.649368	16.23966	.6696675
3	12.31891	-2.003476	10.69861	.9450001
4	12.31891	-3.507318	12.7358	.8315716
5	12.31891	-2.238415	15.33633	.6963639
6	12.31891	-.2994328	10.31397	.9704173
7	12.31891	-1.764905	11.3705	.8958634
8	12.31891	-1.817978	11.81992	.8665543
9	12.31891	-9.240908	10.95218	.9528462
10	12.31891	-5.135808	13.79069	.799715
11	12.31891	-.9275115	11.17482	.9029264
12	12.31891	-3.790026	10.89878	.9388133
13	12.31891	-.03122853	13.91343	.7193599
14	12.31891	-1.830868	10.66952	.9464403
15	12.31891	-3.486403	14.98909	.7299616
16	12.31891	-3.902869	11.41346	.9077152
17	12 [illegible]	[illegible]	11.10271	.890967
18	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible].15207	8.098103	[illegible]	[illegible]
	17.38791	8.218698	20.149[illegible]	[illegible]
42	21.35774	8.367318	25.50865	.09524828
43	12.15207	8.39989	21.75594	.1198041
44	12.15207	8.626873	16.6039	.1721353
45	17.38791	8.818168	23.73837	.0792102
46	21.35774	8.818706	25.25642	.07186482
47	12.15207	9.079551	25.67919	.05544995
48	21.35774	9.23802	20.28676	.06896536
49	22.17403	9.953182	21.11887	.004193038

Figure 6. A CELLSIM position results table generated by the source program shown in Figure 1.

The table contains one row for each group in the state/population. The columns describe the number of cells in the group, the time the group entered the state, the time the group should exit the state, and the fraction of passage through the state completed.

The NOTES of the table contain an INOCULUM command, which you can use to restart the simulation with cells in the same position as reported in the table. As with the other output described above, CELLSIM gives you the option of making a top-level copy of the POSITIONS results table.

When all other results have been presented, CELLSIM asks:

```
DO YOU WISH TO HAVE THE RESULTS PRINTED
  OUT IN ORIGINAL CELLSIM FORMAT?
```

If you answer YES, the results of the simulation, in the line-printer format originally used by the FORTRAN version of CELLSIM, are printed out and mailed to you. This printout may be useful for comparing your results with those obtained on another CELLSIM installation. As the last step in its execution, CELLSIM types out the amount of CPU (computer) time and REAL (clock) time it took for this run. The CPU figure may be useful for estimating batch run-time limits (see below).

Running CELLSIM in Batch Mode

As noted above, after you tell CELLSIM where to get the source program text, it asks:

```
DO YOU WISH TO RUN CELLSIM AS A BATCH JOB?
```

If you answer YES, CELLSIM asks you when (day and time) you want the batch job to run, and how much CPU time to allow for the run. The job will terminate catastrophically if it runs out of time, so it is a good idea to overestimate. The run times indicated for the non-batch CELLSIM runs (see above) are often helpful estimating CPU-time requirements.

When you have specified the parameters for the batch job, CELLSIM queues it up for future execution and types the following message:

```
WHEN YOUR BATCH JOB IS COMPLETED, YOU WILL BE
ABLE TO GET BACK THE RESULTS OF THE CELLSIM
RUN BY ISSUING THE FOLLOWING COMMANDS:

    RAC PUBLIC CELLSIM <GO>
    CALL RBCELLSIM <GO>

WHEN "RBCELLSIM" ASKS FOR A THREE-CHARACTER
PREFIX, YOU SHOULD TYPE:

    xxx <GO>

    CELLSIM WILL THEN PROCEED AS USUAL.
```

A three-letter code such as AAA or AAB appears in the place of "xxx" in the message. Here CELLSIM is handing you a pawn ticket with which you can get back the results of this batch run later. Because each pending batch request receives a unique pawn ticket (three-letter code) you can queue up many CELLSIM jobs for an overnight run. After typing this message, CELLSIM returns control to you.

At the appointed time, the batch job is run. If the batch job fails, you will later receive a PROPHET message (sent from you to yourself) indicating that the batch job failed. If the run is completed, you will receive two messages, one from the batch job manager, and one of the following form:

```
YOUR CELLSIM BATCH JOB (CODE xxx) FINISHED,
TAKING SOMEWHAT MORE THAN nnn MINUTES OF
CPU TIME.
```

The "xxx" is the three-letter code for this run, as described above. The CPU time (in minutes) may help you to estimate bounds for future batch jobs.

Using RBCELLSIM To Retrieve Results of a Batch Run

To retrieve the results of a batch job, invoke the CELLSIM procedure at its alternate entry point, RBCELLSIM (for Read Batch). To do this, you must first restore and compile CELLSIM (unless you have already given the RAC CELLSIM command in this work session). Then type:

```
CALL RBCELLSIM <GO>
```

CELLSIM prompts you to enter the three-letter code assigned when the batch job was submitted. When you type in the code, CELLSIM may respond with the error message:

> THERE DOESN'T APPEAR TO BE A RUN WITH THAT PREFIX. TRY AGAIN.

This batch job error message may be issued if you already picked up the batch results, if the batch job aborted, or if some system failure occurred; please consult your PROPHET representative for help.

If you do not receive this error message, all is well. You are now exactly where you would be in a non-batch run after returning from the FORTRAN program. CELLSIM types out messages for any errors and asks whether you want to see the REPORTS table.

Reference

Donaghey CE. CELLSIM II User's Manual. Department of Industrial Engineering, Cullen College of Engineering, University of Houston, TX, March 1977.

5. Utility
Procedures

5. Utility
Procedures

5. Utility Procedures

COMPSAVED	recompiles saved procedures	5-1
FINDARCHIVED	performs census of archived tables on an archive pack.	5-3
DELETEARCHIVED	sets up a batch job that deletes tables on an archive pack using results of FINDARCHIVED.	5-9
MAKEFORMATTAB	creates a format table to be used by READFILE	5-13
READFILE	transfers data from local computers to PROPHET tables	5-19
READTEXT	transfers data from local computers to PROPHET TEXT variables	5-29
P2MLAB	transfers data from PROPHET to MLAB	5-31
MLAB2P	transfers data from MLAB to PROPHET	5-35
P2BMDP	transfers PROPHET data to BMDP, creates a control file, and runs BMDP programs	5-39

COMPSAVED

by James Jay Wood
Bolt Beranek and Newman Inc.

Summary

COMPSAVED recompiles saved procedures and resaves them in your PRIVATE or SHARABLE SAVED table.

Calling Sequence

```
RAC PUBLIC COMPSAVED <GO>
CALL COMPSAVED <GO>
```

Entry

```
CALL COMPSHARED <GO>
CALL COMPNAMED(named-saved-table)
                <table>
```

Citation

No citation necessary.

Questions or Comments

Contact Jim Wood on the PROPHET line.

Things to Remember

COMPSAVED uses much CPU time. You should use it only when necessary, that is, when a new version of PROPHET necessitates recompiling saved procedures.

When a new version of PROPHET is brought up, you must recompile all of your saved procedures. Recompiling all of your procedures by hand would be a considerable nuisance; the public procedure COMPSAVED makes this job a little easier.

Using COMPSAVED

When you give the command

```
CALL COMPSAVED <GO>
```

all saved procedures in your PRIVATE SAVED table are recompiled and resaved.

To recompile sharable saved procedures, use the entry COMPSHARED. The command is:

```
CALL COMPSHARED <GO>
```

If you have a NAMED saved table, you can use the entry COMPNAMED. The command is:

```
CALL COMPNAMED(named-saved-table)
```

where named-saved-table is the saved table you want to update.

NOTE: Do <u>not</u> use COMPSAVED to collect the garbage in a saved table; use the system command %GARBAGECOLLECT, which is documented in the <u>PROPHET User's Manual</u>.

FINDARCHIVED

by James Jay Wood
Bolt Beranek and Newman Inc.

Summary

FINDARCHIVED sets up a batch job to perform a census of archived tables on an archive pack. The batch job creates a subdirectory table for use with DELETEARCHIVED.

Calling Sequence

```
RAC PUBLIC FINDARCHIVED <GO>
CALL FINDARCHIVED <GO>
```

Entry

```
CALL FINDSHARARCHIVED <GO>
```

Citation

No citation necessary.

Questions or Comments

Contact Jim Wood on the PROPHET line.

Things to Remember

Use FINDARCHIVED only in conjunction with DELETEARCHIVED.

The public procedure FINDARCHIVED enables you to make a subdirectory table containing information about all of your private or sharable tables that are archived on one archive pack. You can then use this subdirectory table to decide which tables you want to keep and which ones you want to delete.

The public procedure DELETEARCHIVED uses the subdirectory created by FINDARCHIVED and, without dearchiving the tables listed in it, deletes them. Because the tables are not dearchived, you can save considerable computer charges.

Calling the Correct Entry

Call FINDARCHIVED if you want a census of PRIVATE tables archived on a given pack. Otherwise, if you want a census of SHARABLE tables archived on a given pack, call the entry point FINDSHARARCHIVED. The only difference between FINDARCHIVED and FINDSHARARCHIVED is that each searches for a different table type (PRIVATE or SHARABLE).

Choosing an Archive Pack

The procedure first asks you to specify the name of the archive pack to search.

Six archive packs are currently used on the PROPHET System: UPBA, UPBB, UPBC, UPBD, UPBE, and the current archive pack, UPBX. UPBA contains the tables that have been archived for the longest period of time, UPBB the next longest, and so on.

The procedure makes a quick check to see whether you have any tables archived on the selected archive pack. If you do not, it informs you of this, and you can call FINDARCHIVED again to search another pack.

If you do have some tables archived on the selected pack, FINDARCHIVED asks whether you want to submit a batch job to perform the census. Normally, you should answer YES.

If you have been using PROPHET for some time and/or have many archived tables, you will probably need to run FINDARCHIVED for several packs. However, you are permitted to use FINDARCHIVED for only one pack per day!

Detailed Table Information

When you call FINDARCHIVED, it asks whether you want a detailed type-out of information about each archived table. If you do, the procedure types out the heading and data type of each column in an archived table as well as any notes associated with the table. Instructions on examining this information are given below (see "Log File of the Batch Census Job").

If you do not choose the detailed type-out, you are given the following information about each archived table:

- number of rows and columns
- number of disk blocks used by the table
- date the table was last accessed

The Batch Census Job

FINDARCHIVED itself does not actually perform the census; it sets up an (unattended) batch job, which begins running sometime after 12 midnight (Eastern Time) on the day you call FINDARCHIVED.

Subdirectory Table

The batch census job looks through your archived tables, locates all of those archived on the selected archive pack, and builds a subdirectory table containing information about each archived table. The information includes the number of disk blocks used by the archived table, the number of rows and columns, and the date you last accessed it. In addition, column 5 of the subdirectory table contains information about the file associated with each table.

The subdirectory table is given the name %<pack-name>TABLES. For example, if you called FINDARCHIVED and requested information on all private tables archived on UPBD, the subdirectory would look like the one in Figure 1. Its name would be %UPBDTABLES.

%UPBDTABLES 6R X 5C

0 TABLES	1 DISK BLOCKS	2 NUM. ROWS	3 NUM. COLS	4 DATE LAST ACCESSED	5 FILE
1. DATA1	10	40	3	9/22/81	UPBD:DATA1.TBL[200,217]
2. DATA2	5	5	4	8/13/81	UPBD:DATA2.TBL[200,217]
3. DATA3	5	5	4	8/13/81	UPBD:DATA3.TBL[200,217]
4. MYPROCS	65	13	5	9/16/81	UPBD:MYPRCS.TBL[200,217]
5. MYTAB				FILE MISSING	UPBD:MYTAB.TBL[200,217]
6. OLDPROCS	15	5	5	9/16/81	UPBD:OLDPRC.TBL[200,217]

Figure 1. Typical subdirectory table created by FINDARCHIVED.

Note in row 5 of Figure 1 that the procedure could not find the table named MYTAB even though the PRIVATE ARCHIVE table listed it as being archived on archive pack UPBD. When the procedure cannot find a table, it informs you of this in column 4 of the subdirectory.

When a cell in column 4 of your subdirectory table contains the advisory message FILE MISSING instead of a date, you should contact your system representative immediately to determine what happened to the missing table.

Log File of the Batch Census Job

If you answered YES to the question in FINDARCHIVED about having a detailed type-out for each archived table, you will need to look at the log file of the batch census job. (Every batch job creates a log file containing a record of the commands the job gave and the system's responses.) The detailed type-out includes the heading and data type of each column of the table, as well as any notes or footnotes associated with the archived table.

To see this more detailed information, you must look at the log file of the batch census job. In fact, it is a good idea to look at this log file whether you requested the detailed type-out or not. Occasional problems arise that could seriously affect the state of your tables.

The simplest way to look at the log file is to use the system procedure $READLOG, which is documented with $AUTOBAT in the PROPHET User's Manual. When you call $READLOG, it asks you for the name of the batch job. This name is always the name of the archive pack selected in FINDARCHIVED.

Thus, to look at the log file of the batch census job that created the example in Figure 1, you would have the following interaction:

```
CALL $READLOG<GO>
WHAT IS THE NAME OF THE JOB
WHOSE LOG FILE YOU WISH TO EXAMINE? UPBL<GO>
WOULD YOU LIKE TO PROCESS AND GET THE LOG
   IMMEDIATELY?  YES<GO>
```

(Your command and responses are underlined.) You should make hard copies of these log files for your records.

Reviewing your Archived Tables

Within a day or two after you use FINDARCHIVED, you should review the tables listed in the subdirectory. If you want to keep all the archived tables named in the subdirectory, nothing remains to be done except to delete the subdirectory table.

Because one purpose of the subdirectory is to provide a list of archived tables for deletion, you need to delete only the row whose column 0 entry is the name of a table you want to preserve on the archive system. Remember that the tables you leave in the subdirectory are the ones that DELETEARCHIVED deletes.

If you need to look at some of the tables listed in the subdirectory, dearchive them and decide what you want to do. (If you do dearchive a table, you should delete the row containing information about it from the subdirectory, because it is no longer archived on that archive pack.)

DELETEARCHIVED

by James Jay Wood
Bolt Beranek and Newman Inc.

Summary

DELETEARCHIVED sets up a batch job that deletes tables on an archive pack. The tables must previously have been gathered into a subdirectory table created by FINDARCHIVED.

Calling Sequence

```
RAC PUBLIC DELETEARCHIVED <GO>
CALL DELETEARCHIVED <GO>
```

Entry

```
CALL DELETESHARARCHIVED <GO>
```

Citation

No citation necessary.

Questions or Comments

Contact Jim Wood on the PROPHET line.

Things to Remember

Use DELETEARCHIVED only in conjunction with FINDARCHIVED.

The public procedure DELETEARCHIVED uses the subdirectory created by FINDARCHIVED and, without dearchiving the tables listed in it, deletes them. Make sure you are familiar with the workings and results of FINDARCHIVED before you use DELETEARCHIVED.

FINDARCHIVED Subdirectory Table

After you use FINDARCHIVED or FINDSHARARCHIVED to build a subdirectory table for a given archive pack, review the tables listed in the subdirectory, and delete the rows of subdirectory containing information about the tables you want to _keep_, you can use the public procedure DELETEARCHIVED to delete the tables remaining in the subdirectory without actually dearchiving them.

DELETEARCHIVED Entries

If you created your subdirectory table using FINDARCHIVED (i.e., if the subdirectory contains private tables), you should use DELETEARCHIVED. If you created your subdirectory table using FINDSHARARCHIVED (i.e., if the subdirectory contains sharable tables), you should use the entry point DELETESHARARCHIVED. Once invoked, DELETEARCHIVED and DELETESHARARCHIVED behave identically. They ask you only for the name of the subdirectory table containing the list of tables archived on a given archive pack.

Like FINDARCHIVED, DELETEARCHIVED submits a batch job that actually does the work.

The Batch Delete Job

There are several points to remember about the batch delete job:

- The batch job deletes tables on only one archive pack. If you dearchive any tables listed in the subdirectory or move them to another archive pack, they cannot be deleted. For this reason, you should run the DELETEARCHIVED procedure _within a few days of_ creating the subdirectory.
- As with FINDARCHIVED, DELETEARCHIVED allows you to submit _only one_ of these batch jobs for a given day.
- Examine the log file of the batch delete job using the procedure %READLOG. (%READLOG is documented in the _Prophet User's Manual_.) The name of the batch delete job is DELJOB. The log file of the batch delete job contains information about any problems it might have encountered. In addition, the batch delete job adds a column to the subdirectory that

contains a short description of what happened for each table. Figure 1 shows how the subdirectory table discussed in FINDARCHIVED looks after the batch delete job is finished.

%UPBDTABLES: DELETEARCHIVED RUN ON 9/25/81 5R X 6C

0 TABLES	1 DISK BLOCKS	2 NUM. ROWS	3 NUM. COLS	4 DATE LAST ACCESSED	5 FILE
1. DATA1	10	40	3	9/22/81	UPBD:DATA1.TBL[200,217]
2. DATA2	5	5	4	8/13/81	UPBD:DATA2.TBL[200,217]
3. DATA3	5	5	4	8/13/81	UPBD:DATA3.TBL[200,217]
4. MYPROCS	65	13	5	9/16/81	UPBD:MYPRCS.TBL[200,217]
5. OLDPROCS	15	5	5	9/16/81	UPBD:OLDPRC.TBL[200,217]

0 TABLES	6 ACTION TAKEN
1. DATA1	DELETED: 9/25/81
2. DATA2	DELETED: 9/25/81
3. DATA3	DELETED: 9/25/81
4. MYPROCS	DELETED: 9/25/81
5. OLDPROCS	DELETED: 9/25/81

Figure 1. Subdirectory table after completion of the batch delete job.

Warning!

Under no circumstances whatsoever should you be running PROPHET when you have set up a batch delete job to run!!! This job is always set up to run sometime after 12 midnight (Eastern Time). Serious problems can arise if you try to run two jobs simultaneously.

MAKEFORMATTAB

by Hank Baig
Bolt Beranek and Newman Inc.

Summary

MAKEFORMATTAB interactively creates a format table to be used by the public procedure READFILE. A format table contains information that tells READFILE how to interpret files that you want it to transfer to PROPHET tables.

Calling Sequence

```
RAC PUBLIC MAKEFORMATTAB <GO>
CALL MAKEFORMATTAB <GO>
```

Citation

No citation necessary.

Questions or Comments

Call Hank Baig on the PROPHET line.

Things to Remember

You should be familiar with the public procedure READFILE before you use MAKEFORMATTAB.

The public procedure MAKEFORMATTAB creates a table that contains information on how to interpret files that you wish to transfer to PROPHET tables using the READFILE public procedure. MAKEFORMATTAB prompts you interactively for all the information required. Keep in mind when reading the following description of format tables that you do not need to keep track of what pieces of information go in which cells of the format table, because MAKEFORMATTAB does this for you.

The Format Table

Basically, a format table has a row entry for each column of the data table. Column 1 of the format table contains the HEADING for each column and column 2 contains the TYPE of the column. This means that there is a row in the format table for each column in the destination data table. The row names of the format table are COLUMN 1, COLUMN 2, and so on. If the data table has row names, the first row in the format table has the row name ROWNAMES.

Reading the File

READFILE reads each line in the file in sequence, but you control where each datum in the line is to be placed. It may be placed in the next row of the same column (termed reading the file "down columns") or in the next column of the same row (reading the file "along rows").

There are two ways to indicate where in the file to move to the next column (when reading down columns) or the next row (when reading along rows):

- specify that the file contains a special "delimiter character" which tells READFILE to move to the next column (or row)
- specify movement to the next column (or row) after a certain number of data values are read

You can also specify multiple delimiter characters or numbers of data values. For example, you might state that there are five rows in column 1, four rows in column 2, and seven rows in column 3 of a particular table.

Reading Down Columns

If the data file is read down columns, column 3 of the format table contains the delimiter character or the number of row entries for each column. If you have only a single delimiter value or a constant number of rows in each column, then MAKEFORMATTAB enters the value only for the first column.

Reading Along Rows

If the data are read along rows, then the format table contains at least one additional row to indicate this. For each row, you may specify either the number of columns in that row or a delimiter character, which is entered in column 3 of the format table. If you wish to specify a constant number of columns per row or only a single delimiter value, then only this one additional row in the format table is needed.

Interacting with MAKEFORMATTAB

When you call MAKEFORMATTAB, the procedure first prompts you for the name you want the format table to have. It then asks whether the destination data table has row names. Remember: if you answer YES, the first row in the format table has the name ROWNAMES.

You are then prompted to enter the number of columns in the data table. You are next asked whether all the columns of the table are of the same type. Type YES (Y) if they are, and the procedure will prompt you for the type. If they are not, you enter the column types when you enter the column headings, as described below. You are then asked whether the file should be read "down columns".

Delimiters

You are then prompted to enter the delimiter value (or number of rows/columns or columns/rows). You may enter a single value, or multiple values on a line separated by spaces.

Note that you can use any character (including carriage return-linefeed) as a delimiter value.

Column Headings

The procedure next prompts you for the heading of each column, and the type (if you have not specified a common type for each column as described above).

Data Separator

READFILE assumes that the data values on a line of the file are separated by spaces or commas, but you can change this. You are asked whether you wish to change the "data separator". If you answer YES (Y), you are asked whether you wish to specify a separator character (or a "field width"). You then specify either the separator character or the list of numbers (separated by spaces) representing the field widths.

The field width enables you to specify that each datum on a line of the file occupies, for example, 6 spaces (including blanks). You can also specify multiple field widths. For example, you could specify that the first value occupies 6 spaces; the second, 3; the third, 8. Note that you may specify a "skip-field" by entering a negative number in the list.

If you do change the data separator to a new character, MAKEFORMATTAB records the new value in footnote 2 of the format table. Field widths are recorded in footnote 3.

EMPTY Character

Finally, the procedure asks whether there is a special character that, when read by READFILE, indicates that that particular cell in the table is EMPTY. If there is such a character, MAKEFORMATTAB records it in footnote 1 of the format table.

Example

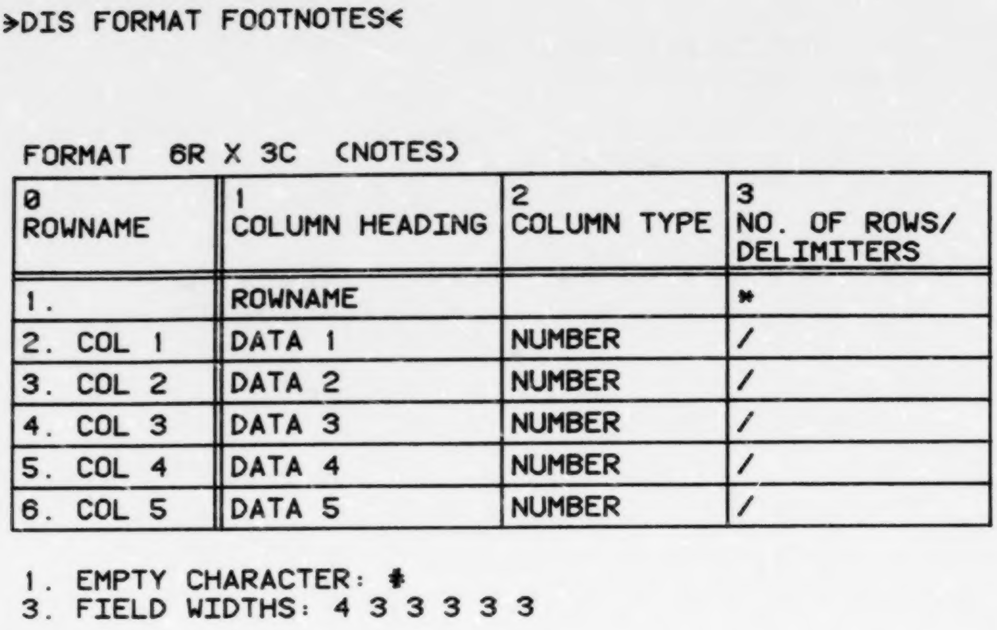
Consider the following sample file:

```
100.      200.
101.      204.
103.      206.
105.      210.
107.      211.
```

These data are to be read down columns into a table of two rows and five columns. You could set up a format table for this file by interacting with MAKEFORMATTAB as shown in Figure 1. Figure 2 depicts the resulting format table.

```
>RAC PUBLIC MAKEFORMATTAB<
>CALL MAKEFORMATTAB<

NAME FOR FORMAT TABLE ?FORMAT<
ARE THERE ROWNAMES ?YES<
NUMBER OF COLUMNS IN DATA FILE ?5<
ARE ALL COLUMNS OF THE SAME TYPE ?YES<
TYPE OF COLUMN ?NUMBER<
WILL YOU READ THE FILE DOWN COLUMNS ?YES<
DO YOU WISH TO SPECIFY A SINGLE DELIMITER
(OR NUMBER OF ROWS/COLUMN)?NO<

DELIMITER VALUE/NUMBER OF ROWS FOR
  ROWNAMES COLUMN ?*<

HEADING OF COLUMN 1?DATA 1<
NUMBER OF ROWS/DELIMITER VALUE?/<

HEADING OF COLUMN 2?DATA 2<
NUMBER OF ROWS/DELIMITER VALUE?/<

HEADING OF COLUMN 3?DATA 3<
NUMBER OF ROWS/DELIMITER VALUE?/<

HEADING OF COLUMN 4?DATA 4<
NUMBER OF ROWS/DELIMITER VALUE?/<

HEADING OF COLUMN 5?DATA 5<
NUMBER OF ROWS/DELIMITER VALUE?/<

SPECIAL "EMPTY" CHARACTER (<GO> IF NONE) ?#<
DO YOU WISH TO CHANGE THE DEFAULT DATA SEPARATOR ?YES<
DO YOU WISH TO SPECIFY DELIMITER
  (AS OPPOSED TO FIELD WIDTH(S))?NO<
ENTER FIELD WIDTH(S) ON A LINE ?4 3 3 3 3 3<
```

Figure 1. Using MAKEFORMATTAB.



```
>DIS FORMAT FOOTNOTES<
```

FORMAT 6R X 3C (NOTES)

0 ROWNAME	1 COLUMN HEADING	2 COLUMN TYPE	3 NO. OF ROWS/ DELIMITERS
1.	ROWNAME		*
2. COL 1	DATA 1	NUMBER	/
3. COL 2	DATA 2	NUMBER	/
4. COL 3	DATA 3	NUMBER	/
5. COL 4	DATA 4	NUMBER	/
6. COL 5	DATA 5	NUMBER	/

1. EMPTY CHARACTER: #
3. FIELD WIDTHS: 4 3 3 3 3 3

Figure 2. A typical format table.

READFILE

by Hank Baig

Bolt Beranek and Newman Inc.

Summary

READFILE is an interface from secondary computers to PROPHET. It enables you to transfer data from local files into PROPHET tables.

Calling Sequence

```
RAC PUBLIC READFILE <GO>
CALL READFILE <GO>
```

Citation

No citation necessary.

Questions or Comments

Contact Hank Baig on the PROPHET line.

Things to Remember

You should read the description of the public procedure MAKEFORMATTAB in this notebook before you try to create a format table within READFILE (Option 2).

The public procedure READFILE is an interface between PROPHET and other computers. It enables you to transfer data from a secondary computer (e.g., a local microcomputer) into PROPHET tables. The process consists of three steps:

(1) specifying the format of the data file and PROPHET table

(2) sending the data file to PROPHET

(3) making and filling the PROPHET table

File Formats

READFILE affords great flexibility in the organization of your data file. Ultimately, the file is read one line at a time, but you have complete control over where each datum in the file is placed in the table. For example, you can specify that each successive datum on a line be placed in the next row in the current column. This is termed reading "down columns". Alternatively, you can specify that each successive datum on a line be placed in the next column in the current row. This is termed reading "along rows".

To read data down columns, specify any of the following:

- a constant number of rows to fill in each column
- different numbers of rows for each column in the table
- a special delimiter character that, when read by READFILE, causes subsequent data to be placed into the next column (beginning at row 1)
- separate delimiters for each column.

Four analogous methods of incrementing to the next row of the table are available when you read the data "along rows".

Restrictions and Error-Checking

There are a few restrictions on the contents and form of the data file. The data file must consist of lines of ASCII data. The only types of data permitted are TEXT and NUMBER.

You can add error-checking information to a file by running the BASIC program ERRCHK on your local machine. ERRCHK adds three fields to each line: a line number, a checksum, and the number of characters on the line.

Specifying the File Format

When you call READFILE, the procedure first prompts you to enter the method of specifying the format of the data files. It gives you four options:

(1) Use a format table
(2) Create a format table
(3) Use a template table
(4) Simple specification

READFILE enables you to use a format table to specify the the manner in which the file is read into the data table. A format table is particularly useful when you have several files with a similar format.

Before calling READFILE, you can use the public procedure MAKEFORMATTAB (described in this notebook) to make format tables of the correct form; you can also create a format table within READFILE. Specify Option 1 if you want to use an existing format table, Option 2 if you want to create one.

Choose Option 3 when you have a data table whose column headings and types are to be used in the new PROPHET data table. READFILE asks for the name of the template table and whether the file should be read down columns.

It next asks you to enter delimiters or numbers of rows per column (or columns per row). You may specify either a single value for the delimiter (or columns/row or rows/column) or multiple values, separated by spaces, on a line.

Option 4, simple specification, enables you to specify all the format information needed. READFILE asks for the number of columns in the data table, whether the file should be read down columns, and for the information about single or multiple delimiters.

Number of Rows in the Data Table

If you do not explicitly provide the number of rows in the data table, the procedure tries to estimate it from the other information you provide.

If READFILE can make an estimate, it reports this number and asks whether it is correct. If you respond that it is not, READFILE asks whether you want to respecify. If you do not respecify, the procedure reads the file a preliminary time to calculate the number of rows.

If READFILE cannot estimate the number of rows, it also asks you whether you want to specify this number. If you do not, it makes a preliminary pass to calculate the number.

Specifying the Data Separator Character

If you are using Option 3 or 4, READFILE asks whether you wish to change the default value that separates individual data in the file. The default is spaces or commas between data.

If you do want to change this value, you can enter any character you choose. Suppose, for example, that you have data consisting of first and last names separated by a comma and a space. You could separate the individual data by a different character, such as a tab.

You also have the option of supplying a "field width" instead of a new separator character. This option enables you to specify that each datum on a line of the file occupies, for example, 6 spaces (including blanks). You can also specify multiple field widths. For example, you could specify that the first datum occupies 6 spaces; the second, 3; the third, 8. (A negative field width indicates skipping that number of characters.)

READFILE asks whether you wish to specify a special character to represent EMPTY. If you do, you must enter it. If you do not want to specify a special character, simply press <GO>.

Naming the Data Tables

READFILE then asks for the names of the data tables. Note that all of these tables and all of the files to be read into them must have the format you specified as described above.

You enter the table names on a line, separated by spaces or commas. The first set of file data is now ready to be transmitted.

Transmitting the Data

READFILE signals that it is ready to transmit your file data by typing the message

```
READY TO PROCESS TABLE data-table TRANSMIT
  (END WITH CTRL-Z <CR> <CR>)
```

where data-table is the name of the first PROPHET table you specified. You should take whatever steps are appropriate for your terminal and modem to send your data file to the NIH computer.

Terminate the transmission by typing

```
CTRL-Z <CR> <CR>
```

where <CR> is carriage return.

Finally, READFILE asks whether your file has checksum information (added by the ERRCHK program). The file is now processed, the table is created, and the data are read into it. If READFILE encounters any errors in reading the data, it reports the line number and the position in the line of the datum in question.

READFILE repeats this transmission process for all tables whose names you specified.

Consult the following examples for further illustration.

```
>CALL READFILE<

        1. USE A FORMAT TABLE
        2. CREATE A FORMAT TABLE
        3. USE A TEMPLATE TABLE
        4. SIMPLE SPECIFICATION
?1<
ENTER NAME OF FORMAT TABLE ?FORMAT<

DO YOU WISH TO SPECIFY THE NUMBER OF ROWS ?NO<
ENTER LIST OF TABLE NAMES (SEPARATED BY SPACES
  OR COMMAS) ?MYTAB<

READY TO PROCESS TABLE MYTAB TRANSMIT
  (END WITH CTRL-Z <CR> <CR>)
>Z^


[TRANSMISSION DONE]
DOES THE FILE CONTAIN CHECKSUM INFORMATION ?NO<
PROCESSED 2 ROWS
```

Figure 1. Using an existing format table.

```
>CALL READFILE<

        1. USE A FORMAT TABLE
        2. CREATE A FORMAT TABLE
        3. USE A TEMPLATE TABLE
        4. SIMPLE SPECIFICATION
?3<
ENTER TEMPLATE TABLE NAME ?TEMPLATE<
IS FILE TO BE READ 'DOWN COLUMNS' ?NO<
DO YOU WISH TO SPECIFY MULTIPLE DELIMITERS
  OR NUMBERS ?YES<
NUMBER OF ROWS ?4<

        ENTER DELIMITER FOR ROW 1?!<
        ENTER DELIMITER FOR ROW 2?!<
        ENTER DELIMITER FOR ROW 3?*<

DO YOU WISH TO CHANGE THE DEFAULT DATA
  SEPARATOR ?YES<
IS THERE A SEPARATOR CHARACTER (AS OPPOSED TO
  FIELD WIDTH) ?YES< ENTER SEPARATOR ?:<

SPECIAL 'EMPTY' CHARACTER (<GO> IF NONE) ?<
ENTER LIST OF TABLE NAMES (SEPARATED BY SPACES
  OR COMMAS) ?DATA1<

READY TO PROCESS TABLE DATA1 TRANSMIT
  (END WITH CTRL-Z <CR> <CR>)
>Z^


[TRANSMISSION DONE]
DOES THE FILE CONTAIN CHECKSUM INFORMATION ?NO<
PROCESSED 4 ROWS
```

Figure 2. Using a template table.

```
>CALL READFILE<

        1. USE A FORMAT TABLE
        2. CREATE A FORMAT TABLE
        3. USE A TEMPLATE TABLE
        4. SIMPLE SPECIFICATION
?4<
ARE THERE ROWNAMES ?NO<
NUMBER OF COLUMNS IN DATA TABLE ?3<
ARE ALL COLUMNS OF THE SAME TYPE ?NO<

        HEADING FOR COLUMN 1?COL1<
        TYPE OF COLUMN 1?NUMBER<
        HEADING FOR COLUMN 2?COL2<
        TYPE OF COLUMN 2?NUMBER<
        HEADING FOR COLUMN 3?COL3<
        TYPE OF COLUMN 3?TEXT<

IS FILE TO BE READ "DOWN COLUMNS" ?YES<
DO YOU WISH TO SPECIFY MULTIPLE DELIMITERS OR
  NUMBERS ?NO<
DELIMITER VALUE OR ROWS/COLUMN ?/<
DO YOU WISH TO SPECIFY THE NUMBER OF ROWS ?NO<
DO YOU WISH TO CHANGE THE DEFAULT DATA
  SEPARATOR ?NO<
SPECIAL "EMPTY" CHARACTER (<GO> IF NONE) ?<
ENTER LIST OF TABLE NAMES (SEPARATED BY SPACES
  OR COMMAS) ?T1 T2<

READY TO PROCESS TABLE T1 TRANSMIT
  (END WITH CTRL-Z <CR> <CR>)
>32 56 81 / 14 / MAX POS FLOAT /
Z^


[TRANSMISSION DONE]
DOES THE FILE CONTAIN CHECKSUM INFORMATION ?NO<
PROCESSED 3 ROWS

READY TO PROCESS TABLE T2 TRANSMIT
  (END WITH CTRL-Z <CR> <CR>)
>53 / 21 77 12 56 / ARG NEG /
^Z


[TRANSMISSION DONE]
DOES THE FILE CONTAIN CHECKSUM INFORMATION ?NO<
PROCESSED 4 ROWS>
```

Figure 3. Specifying all format information.

T1 3R X 3C

	1 COL1	2 COL2	3 COL3
1	32.	14.	MAX
2	56.		POS
3	81.		FLOAT

T2 4R X 3C

	1 COL1	2 COL2	3 COL3
1	53.	21.	ARG
2		77.	NEG
3		12.	
4		56.	

Figure 4. Tables containing transferred file data.

READTEXT

by Hank Baig
Bolt Beranek and Newman Inc.

Summary

READTEXT is an interface between your local computer and PROPHET. It transfers a file from your computer into a TEXT variable in PROPHET.

Calling Sequence

```
RAC PUBLIC READTEXT <GO>
text-var = READTEXT
```

Citation

No citation necessary.

Questions or Comments

Contact Hank Baig on the PROPHET line.

Things to Remember

SAVE text-var before you log out, or it will disappear.

The public procedure READTEXT transfers a file from your local computer (e.g., microcomputer) into a text variable in your PROPHET workarea.

Using READTEXT

To read a file into a TEXT variable text-var, type:

```
RAC PUBLIC READTEXT <GO>
text-var = READTEXT <GO>
```

Text-var will be a TEXT variable containing the contents of the file that you send.

READTEXT asks whether your file contains error-checking information as added by the ERRCHK program (see the description of the public procedure READFILE in this notebook). The procedure then prompts you for the appropriate sequence of commands to send the file.

As usual with TEXT variables in PROPHET, to keep text-var after you log out, you must first SAVE it.

P2MLAB

by Harold Perry
Bolt Beranek and Newman Inc.

Summary

P2MLAB transfers PROPHET data to an MLAB save file and then optionally runs MLAB.

Calling Sequence

```
RAC PUBLIC P2MLAB <GO>
CALL P2MLAB <GO>
```

Entry

```
CALL P2MLABLQT(var-names, MLAB-save-file)
                 <text>        <text>
                   or
               <text array>
```

Citation

No citation necessary.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things to Remember

P2MLAB can only transfer PROPHET data of the following types:

- TABLEs
- FLOAT or FIXED ARRAYs
- FLOAT or FIXED variables

You should be familiar with MLAB before you use P2MLAB.

The public procedure P2MLAB is an interface from PROPHET to MLAB. It transfers PROPHET data to an MLAB save file and then enables you to run MLAB.

Transferring Information to MLAB

When you call P2MLAB, it asks whether you want to name the MLAB save file that will be created for subsequent MLAB use. If you do not specify a name for the save file, P2MLAB automatically names the file MLASAV.

Next, the procedure asks you to type in the names of the variables you want to send to MLAB. Enter each variable as prompted. When you have finished designating the PROPHET variables you want in MLAB, answer the prompt with EXIT.

Running MLAB

You are then asked whether you want to run MLAB. If you do, you leave PROPHET and are sent to MLAB and the variables you just listed for transfer to MLAB are available in MLAB. If you do not elect to run MLAB right after calling P2MLAB, you may retrieve the variables listed for transfer by issuing the "USE name-of-MLAB-save-file" command when you do run MLAB.

Quiet Entry P2MLABQT

You can transfer data to MLAB quietly by using the P2MLABQT entry. P2MLABQT does not ask you for any information.

A call to P2MLABQT requires two arguments:

- var-names, which must be of type TEXT or TEXT ARRAY. This argument specifies the names of the variables you want to send to MLAB.
- MLAB-save-file, which must be of type TEXT. This argument specifies the name of the MLAB save file. If this argument is empty, the procedure uses MLASAV as the MLAB file name.

If you use P2MLABQT, MLAB is invoked automatically and the PROPHET data specified in the call to P2MLABQT is available.

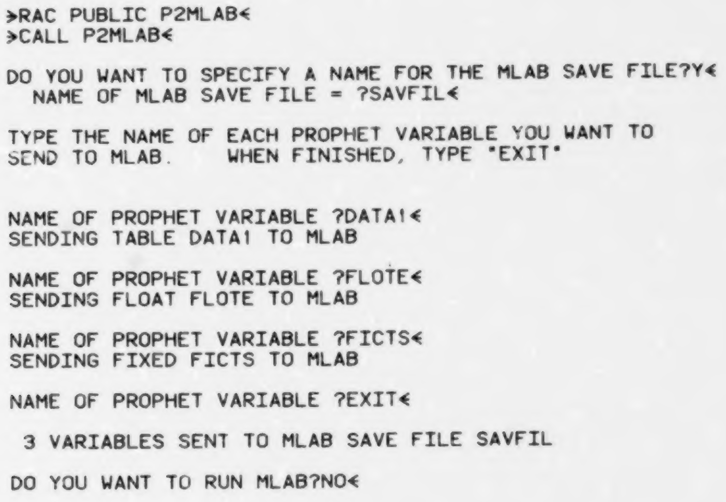
Deleting MLAB Save Files

After you finish with an MLAB save file, you are encouraged to delete, it because these files tend to accumulate and they can occupy considerable space. Assuming that you used P2MLAB or P2MLABQT to create such an MLAB save file with name <filename>, you can delete that file by issuing the following PROPHET command:

DELETE FILE filename.DAT

For example, if in the call to P2MLAB you do not specify a name for the MLAB save file, P2MLAB names that file MLASAV. You can subsequently delete this file by using the PROPHET command

DELETE FILE 'MLASAV.DAT' <GO>



```
>RAC PUBLIC P2MLAB<
>CALL P2MLAB<

DO YOU WANT TO SPECIFY A NAME FOR THE MLAB SAVE FILE?Y<
  NAME OF MLAB SAVE FILE = ?SAVFIL<

TYPE THE NAME OF EACH PROPHET VARIABLE YOU WANT TO
SEND TO MLAB.    WHEN FINISHED, TYPE "EXIT"


NAME OF PROPHET VARIABLE ?DATA1<
SENDING TABLE DATA1 TO MLAB

NAME OF PROPHET VARIABLE ?FLOTE<
SENDING FLOAT FLOTE TO MLAB

NAME OF PROPHET VARIABLE ?FICTS<
SENDING FIXED FICTS TO MLAB

NAME OF PROPHET VARIABLE ?EXIT<

 3 VARIABLES SENT TO MLAB SAVE FILE SAVFIL

DO YOU WANT TO RUN MLAB?NO<
```

Figure 1. Using P2MLAB.

MLAB2P

by Harold Perry
Bolt Beranek and Newman Inc.

Summary

MLAB2P transfers MLAB data to PROPHET.

Calling Sequence

```
RAC PUBLIC MLAB2P <GO>
CALL MLAB2P <GO>
```

Entry

```
CALL MLAB2PQT(MLAB-save-file, tables-flag)
                  <text>          <bool>
```

Citation

No citation necessary.

Questions or Comments

Contact Harold Perry on the PROPHET line.

Things to Remember

MLAB can only transfer MLAB data of the following MLAB types:

- vectors
- matrices
- scalars

You must have used the MLAB "SAVE...IN" command to create the MLAB save file. (You should be familiar with MLAB before you use MLAB2P.)

MLAB2P is designed to transfer data from an MLAB save file to PROPHET. It converts:

- MLAB vectors to PROPHET FLOAT ARRAYs or TABLEs
- MLAB matrices to PROPHET FLOAT ARRAY ARRAYs or TABLEs
- MLAB scalars to PROPHET FLOATs

Interaction with MLAB2P

When you call MLAB2P, the procedure asks you to specify the name of the MLAB save file you want to transfer into PROPHET. (Note that for MLAB2P to work, you must have used the MLAB "SAVE IN file-name" command, where file-name is the name you want the MLAB save file to have.)

The procedure then asks you whether all MLAB vectors and matrices in the specified file should be converted into tables when transferred to PROPHET. MLAB2P goes through the designated MLAB file and converts the MLAB data to PROPHET data. It announces the conversion of each MLAB variable.

After MLAB2P has transformed all MLAB variables to PROPHET, it asks whether you want to delete the MLAB save file. You are encouraged to do so, because these files tend to accumulate and can occupy considerable space.

Quiet Entry MLAB2PQT

You can quietly transfer MLAB data to PROPHET by using the MLAB2PQT entry. MLAB2PQT does not ask you for any information. A call to MLAB2PQT requires two arguments:

- MLAB-save-file, which must be of type TEXT. This argument specifies the name of the MLAB save file you wish to transfer to PROPHET.
- tables-flag, which must be of type BOOL. This argument specifies whether you want MLAB vectors and matrices to be converted to PROPHET tables. If tables-flag has value TRUE, then all MLAB vectors and matrices are converted to PROPHET tables.

MLAB2PQT does not ask whether you want to delete the MLAB save-file. You are encouraged to do so (when these files are no longer useful) by using the command:

```
DELETE FILE file-name
```

Consult Figure 1 for a further description.

```
>RAC PUBLIC MLAB2P<
>CALL MLAB2P<

NAME OF THE MLAB FILE YOU WANT TO BRING INTO PROPHET?SAVFIL<

DO YOU WANT ALL MLAB ARRAYS TURNED INTO PROPHET TABLES?YES<

MLAB VARIABLE: "DATA1" TO PROPHET AS TABLE: "DATA1"

MLAB VARIABLE: "FLOTE" TO PROPHET AS FLOAT: "FLOTE"

MLAB VARIABLE: "FICTS" TO PROPHET AS FLOAT: "FICTS"

1 TABLE BROUGHT INTO PROPHET
DELETE SAVFIL.DAT?YES<
>
```

Figure 1. Using MLAB2P.

P2BMDP

by
Margaret Reid-Miller
Bolt Beranek and Newman Inc.

Summary

P2BMDP is an interface to the BMDP biostatistical package. P2BMDP makes it easy to convert tables to a form that the BMDP programs can use. Just by adding the appropriate BMDP control statements, you can run specific BMDP programs without having to learn the PDP-10 monitor commands. With several P2BMDP commands, you can send PROPHET data to BMDP, create a control file for the data, and run BMDP programs from PROPHET.

Calling Sequence

```
RAC PUBLIC P2BMDP <GO>
CALL P2BMDP <GO>
```

Citation

No citation necessary.

Questions or Comments

Contact Margaret Reid-Miller on the PROPHET line.

Things to Remember

You should be familiar with BMDP before calling P2BMDP.

BMDP files take up a large amount of space, so you should delete them whenever they become unnecessary. However, you must call P2BMDP to delete BMDP files; you cannot access them directly through PROPHET.

BMDP is a biostatistical package interfaced with the PROPHET system. The public procedure P2BMDP is the PROPHET interface to this package. It has several commands that enable you to send PROPHET data to BMDP, create a control file for the data, and run BMDP programs from PROPHET.

P2BMDP Commands

When you call P2BMDP, it asks you the following:

```
Do you want to
1) Make a new control/data file set?    (M)
2) Change an old control file?          (C)
3) Run a BMDP program?                  (R)
4) Delete a set of files?               (D)
5) Scan a list of your BMDP files?      (S)
6) Exit this program?                   (E)
7) Type this text?                      (?)

M,C,R,D,S,E, or ?
```

Each command executes independently, so you can make several input files at once, use any of them as input to a BMDP program, delete those you do not want, or keep them for another PROPHET session. Each command is described in detail below, and a sample session follows.

M (MAKE) Command

Typing M creates a pair of BMDP files from a PROPHET table. The procedure first asks you for the name of your source table. It then asks whether you want to perform an analysis on your entire table; if you answer NO, the procedure prompts you for the rows and columns you want deleted. Non-numeric columns in your table will be ignored, so you need not explicitly exclude them.

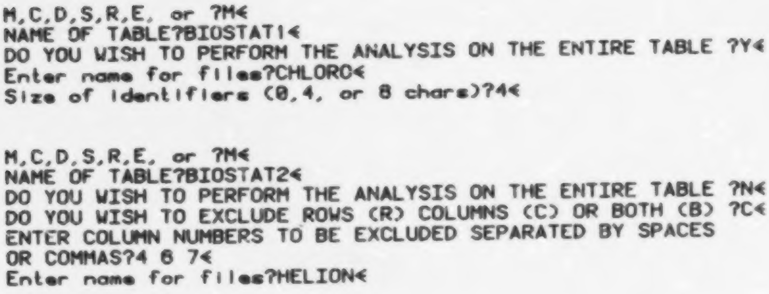
P2BMDP next asks you to supply a name for the files. This name becomes the monitor name for the files that BMDP creates, so it must be one to six alphanumeric characters. If a file already exists with that name, the program asks you for another.

P2BMDP creates two files, a control file containing the control statements expected by BMDP and a separate data file containing the actual numbers from the table. The control file has a monitor name with an extension of ".BMD" and has a skeleton set of BMDP control statements. The BMDP problem takes its title from the table title or, if the table is untitled, from the table name. The variable names are taken from the first eight characters in the column headings; P2BMDP makes them unique by numbering them when necessary.

If the table has row names, you can use the first four or eight characters to identify individual rows for BMDP programs that use them. If the row names are present, the procedure asks whether you want to use zero, four, or eight characters for identifiers. These characters will be ignored by BMDP programs that do not need them, so including them is optional.

The filename of the second file is the same as that of the control file, except that an extension of ".DAT" is added to make the second filename unique and to suggest the contents: the second file contains the actual data in the table, in a suitable format described by the control file. Empty cells in the source table are represented as "9999999." in the data file.

See Figure 1 for two illustrations of the M command.



```
M,C,D,S,R,E, or ?M<
NAME OF TABLE?BIOSTAT1<
DO YOU WISH TO PERFORM THE ANALYSIS ON THE ENTIRE TABLE ?Y<
Enter name for files?CHLORO<
Size of identifiers (0,4, or 8 chars)?4<

M,C,D,S,R,E, or ?M<
NAME OF TABLE?BIOSTAT2<
DO YOU WISH TO PERFORM THE ANALYSIS ON THE ENTIRE TABLE ?N<
DO YOU WISH TO EXCLUDE ROWS (R) COLUMNS (C) OR BOTH (B) ?C<
ENTER COLUMN NUMBERS TO BE EXCLUDED SEPARATED BY SPACES
OR COMMAS?4 6 7<
Enter name for files?HELION<
```

Figure 1. Making new control/data file sets.

C (Change) Command

After P2BMDP creates a set of input files, you can use one of the two PROPHET text editors (see the PROPHET User's Manual) to change BMDP control statements in the control file. When you are asked, type in the identifier (one to six characters) for the control file you want to edit. You can perform many different analyses on the same set of data simply by changing the appropriate commands in the command file for each different BMDP program.

R (Run) Command

To make using BMDP as simple as possible, the procedure runs the BMDP program you specify from PROPHET. Type R to run the program. P2BMDP asks you for the name of the control/data files you want to use and then verifies that they exist.

After you specify the program to run, P2BMDP starts it. Note that you must add the prefix BMDP to the name of the program (e.g., BMDP2V).

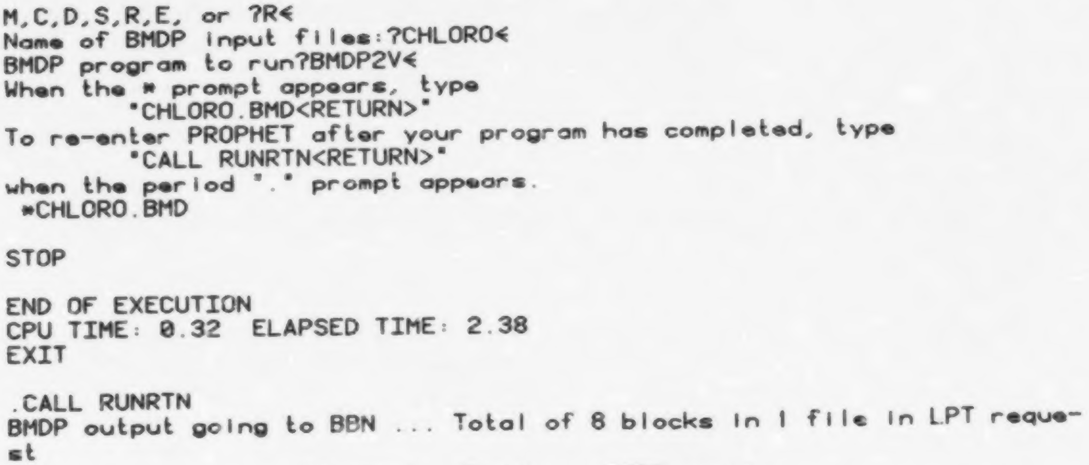
When the BMDP program prints an asterisk prompt (*), you must type the name of the command file followed by <GO>. (This is the only time you will need to use the .BMD extension.) PROPHET then hands control to the BMDP program.

Note that when the program finishes, you are at the monitor, not in PROPHET. To return to PROPHET and P2BMDP, type:

```
CALL RUNRTN <GO>
```

P2BMDP sends the output to the lineprinter, because there is no way to make it readable at the terminal. You must wait for the output to be mailed to you.

Figure 2 illustrates the use of the R command.



```
M,C,D,S,R,E, or ?R<
Name of BMDP input files:?CHLORO<
BMDP program to run?BMDP2V<
When the * prompt appears, type
        "CHLORO.BMD<RETURN>"
To re-enter PROPHET after your program has completed, type
        "CALL RUNRTN<RETURN>"
when the period "." prompt appears.
 *CHLORO.BMD

STOP

END OF EXECUTION
CPU TIME: 0.32  ELAPSED TIME: 2.38
EXIT

.CALL RUNRTN
BMDP output going to BBN ... Total of 8 blocks in 1 file in LPT reque-
st
```

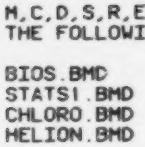
Figure 2. Running a BMDP program.

D (Delete) Command

When BMDP finishes with a set of files, it does not automatically delete them because you may want to run the same data again. You can use the D command to remove the files you no longer need. The files may take up a large amount of space, so you should delete any unnecessary files. Note that the D command deletes both the command file and the data file, because they are closely related.

S (Scan) Command

The S command helps you keep track of your BMDP input files. Type S to display a list of them, as shown in Figure 3.



```
M,C,D,S,R,E, or ?S<
THE FOLLOWING FILES ARE AVAILABLE:

BIOS.BMD
STATS1.BMD
CHLORO.BMD
HELION.BMD
```

Figure 3. Scanning a list of your files.

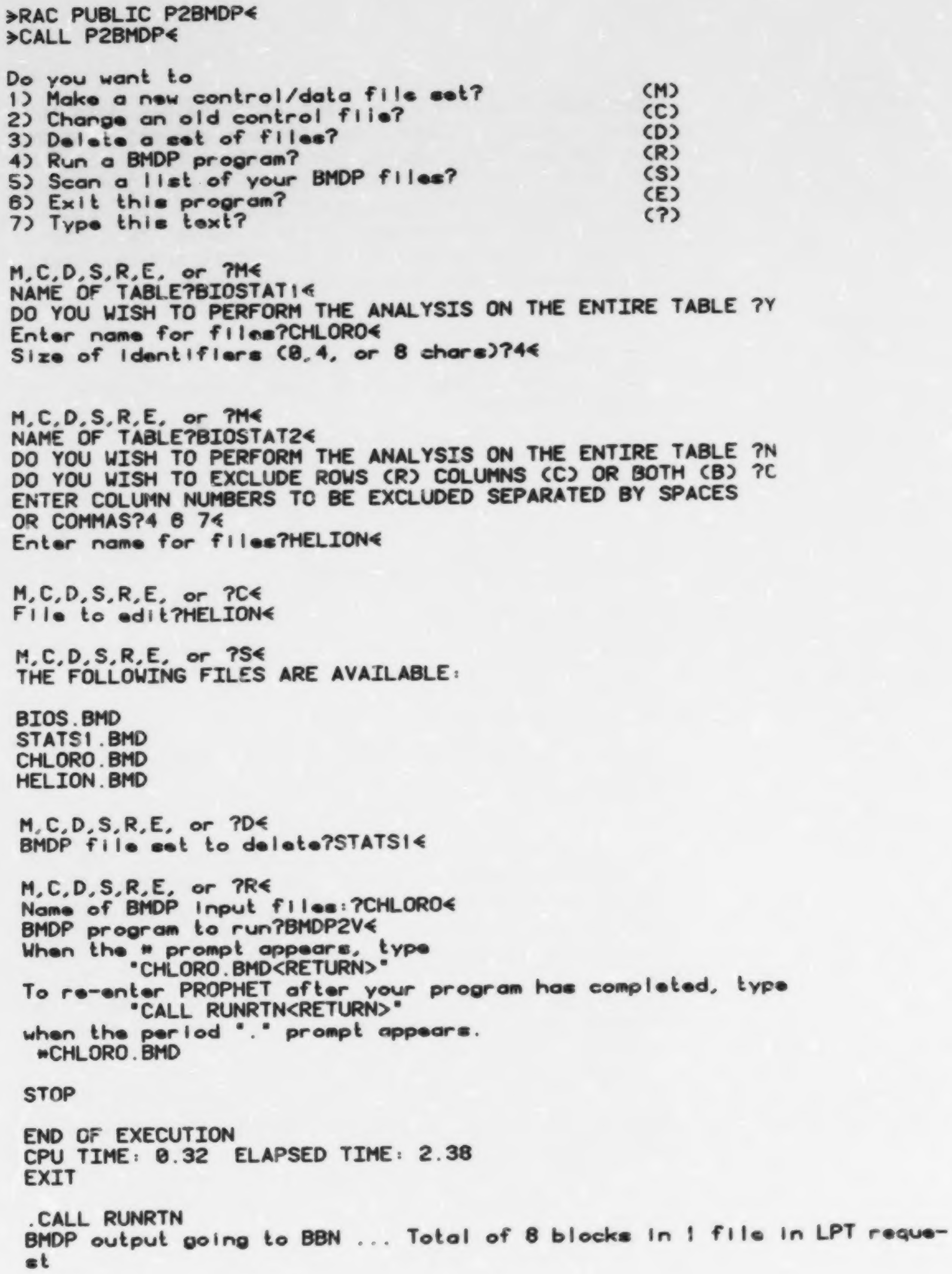
E (Exit) Command

Use the E command to exit P2BMDP and return to top-level PROPHET. Using the E command has no effect on your BMDP input files; they still exist for future sessions. To delete them, you must use the D command before leaving P2BMDP.

You <u>cannot</u> ARCHIVE, SAVE, or DELETE your BMDP files because they are not directly accessible through PROPHET.

? (Menu) Command

Type ? to redisplay the list of P2BMDP functions available.



```
>RAC PUBLIC P2BMDP<
>CALL P2BMDP<

Do you want to
1) Make a new control/data file set?               (M)
2) Change an old control file?                      (C)
3) Delete a set of files?                           (D)
4) Run a BMDP program?                              (R)
5) Scan a list of your BMDP files?                  (S)
6) Exit this program?                               (E)
7) Type this text?                                  (?)

M,C,D,S,R,E, or ?M<
NAME OF TABLE?BIOSTAT1<
DO YOU WISH TO PERFORM THE ANALYSIS ON THE ENTIRE TABLE ?Y
Enter name for files?CHLORO<
Size of identifiers (0,4, or 8 chars)?4<


M,C,D,S,R,E, or ?M<
NAME OF TABLE?BIOSTAT2<
DO YOU WISH TO PERFORM THE ANALYSIS ON THE ENTIRE TABLE ?N
DO YOU WISH TO EXCLUDE ROWS (R) COLUMNS (C) OR BOTH (B) ?C
ENTER COLUMN NUMBERS TO BE EXCLUDED SEPARATED BY SPACES
OR COMMAS?4 6 7<
Enter name for files?HELION<


M,C,D,S,R,E, or ?C<
File to edit?HELION<

M,C,D,S,R,E, or ?S<
THE FOLLOWING FILES ARE AVAILABLE:

BIOS.BMD
STATS1.BMD
CHLORO.BMD
HELION.BMD

M,C,D,S,R,E, or ?D<
BMDP file set to delete?STATS1<

M,C,D,S,R,E, or ?R<
Name of BMDP input files:?CHLORO<
BMDP program to run?BMDP2V<
When the * prompt appears, type
        "CHLORO.BMD<RETURN>"
To re-enter PROPHET after your program has completed, type
        "CALL RUNRTN<RETURN>"
when the period "." prompt appears.
 *CHLORO.BMD

STOP

END OF EXECUTION
CPU TIME: 0.32  ELAPSED TIME: 2.38
EXIT

.CALL RUNRTN
BMDP output going to BBN ... Total of 8 blocks in 1 file in LPT reque-
st
```

Figure 4. A sample P2BMDP session.

6. Fun Procedures

6. Fun Procedures

6. Fun Procedures

HAPPY cheers you up 6-1

POLYGON a many-sided procedure 6-3

SPIRO makes pretty pictures 6-5

LETTERS your own personal calligrapher. 6-9

HAPPY

by Michael Cory
Life Sciences Division
Stanford Research Institute
and Frederick N. Webb
Bolt Beranek and Newman Inc.

Summary

HAPPY provides a glimmer of joy for an otherwise drab day.

Calling Sequence

```
RAC PUBLIC HAPPY <GO>
CALL HAPPY <GO>
```

Things to Remember

Laugh and the world laughs with you. Cry and you cry alone.

HAPPY works (with no grumbling) on all types of terminals.

POLYGON

by Charlotte Hollister
Bolt Beranek and Newman Inc.

Summary

POLYGON draws and prints an n-sided polygon with an optional message centered in it.

Calling Sequence

```
RAC PUBLIC POLYGON <GO>
CALL POLYGON(sides,  X-ctr,  Y-ctr,  diam,  message)
            <fixed> <float> <float> <float> <float>
```

Citation

No citation necessary.

Things to Remember

It is a good idea to issue an "EA" command before calling POLYGON.

POLYGON works <u>only</u> on display terminals.

The public procedure POLYGON draws and prints an n-sided polygon. If desired, you can center a message in the polygon.

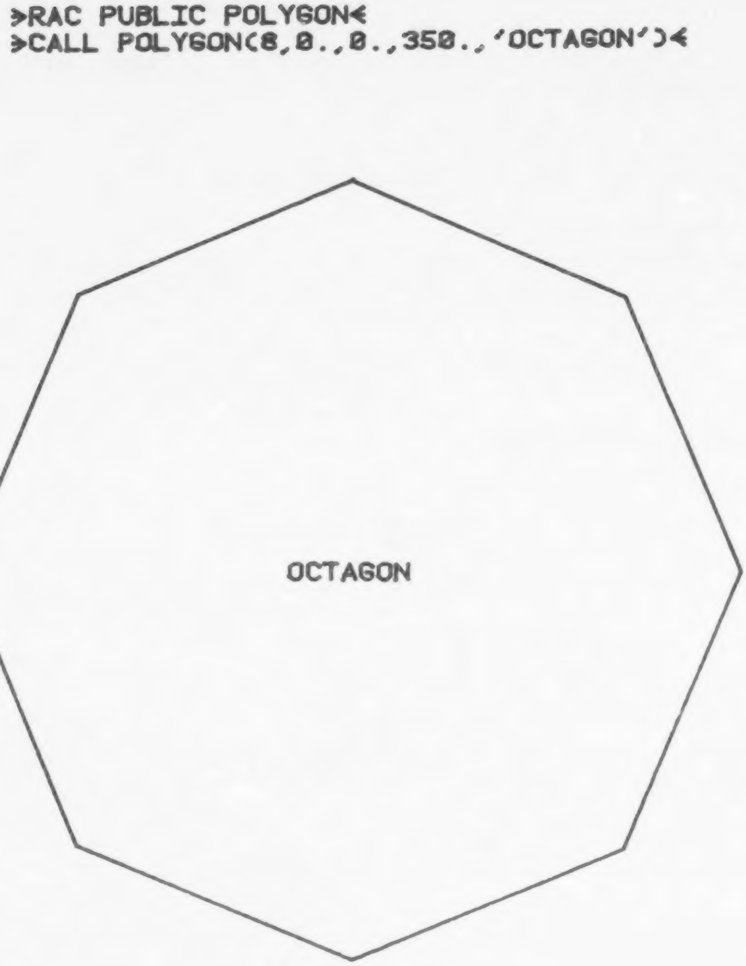
Using POLYGON

To use POLYGON, type

```
CALL POLYGON(sides, X-ctr, Y-ctr, diam, message)
```

where sides is the FIXED number of sides desired; X-ctr is the FLOAT X screen-coordinate of the polygon's center; Y-ctr is the FLOAT Y screen-coordinate of its center; diam is the FLOAT diameter of a circle circumscribing the polygon; and message is a TEXT that, if not empty, will be displayed centered in the polygon.

After POLYGON has finished displaying the polygon and the optional message, it automatically prints a copy of it. See Figure 1 for an example of what POLYGON can do.



```
>RAC PUBLIC POLYGON<
>CALL POLYGON(8,0.,0.,350.,'OCTAGON')<
```

Figure 1. Using POLYGON.

SPIRO

by Mark Lavin
Bolt Beranek and Newman Inc.

Summary

SPIRO draws pretty spirographic pictures.

Calling Sequence

```
RAC PUBLIC SPIRO <GO>
CALL SPIRO(radius,  angle,  scalefac,  nlines)
          <float>  <float>  <float>     <fixed>
```

Citation

No citation necessary.

Things to Remember

Issue the "EA" command before calling SPIRO to prevent overlap of pictures.

SPIRO works only on display terminals.

SPIRO draws pictures like those shown in Figures 1 and 2. You can be creative and try various combinations of numeric arguments, or you can attempt to be systematic by observing how SPIRO works.

How SPIRO Works

SPIRO draws pictures by transforming a point in the plane of the display screen, drawing lines between successive point locations. The transformation consists of a rotation about the origin, and a scaling about the origin. That is, when you say

```
CALL SPIRO(radius, angle, scalefac, nlines)
```

SPIRO first positions the point at radius units along the X-axis. The screen's coordinates are

$500 \leq X \leq +500$,
$390 \leq Y \leq +390$.

On each successive iteration, the point is rotated angle degrees counterclockwise about the origin (screen center), and is then scaled (moved radially) by a factor of scalefac. If scalefac is greater than 1, the curve spirals outward; if scalefac is less than 1 it spirals inward (values in the range .98 to 1.02 are usually good). SPIRO iterates until it has drawn nlines lines.

Have fun!

```
>RAC PUBLIC SPIRO<
>CALL SPIRO(20., 252., 1.02, 150)<
```

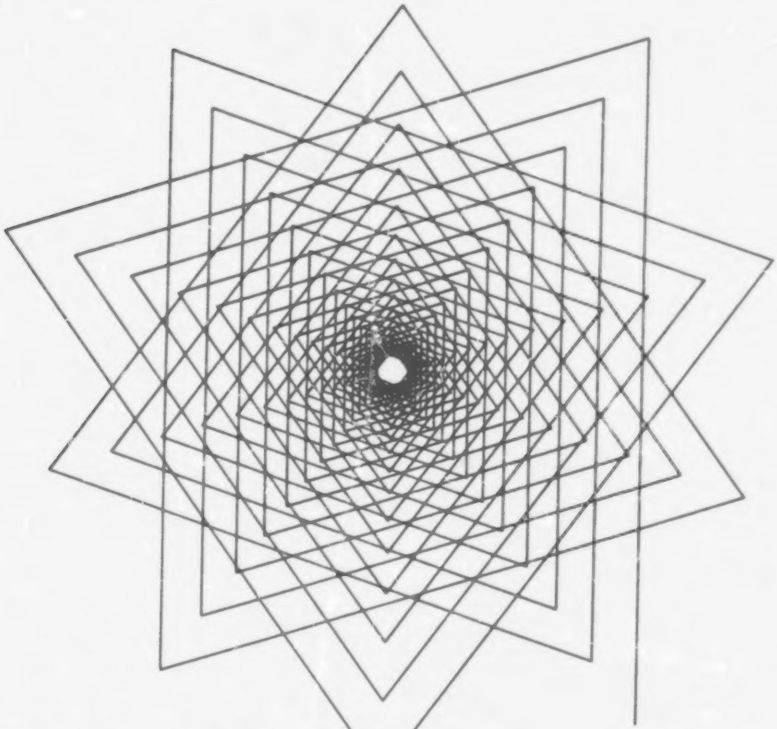


Figure 1. Outward-spiralling design.

```
>RAC PUBLIC SPIRO<
>CALL SPIRO(300., 174., .98, 100)<
```

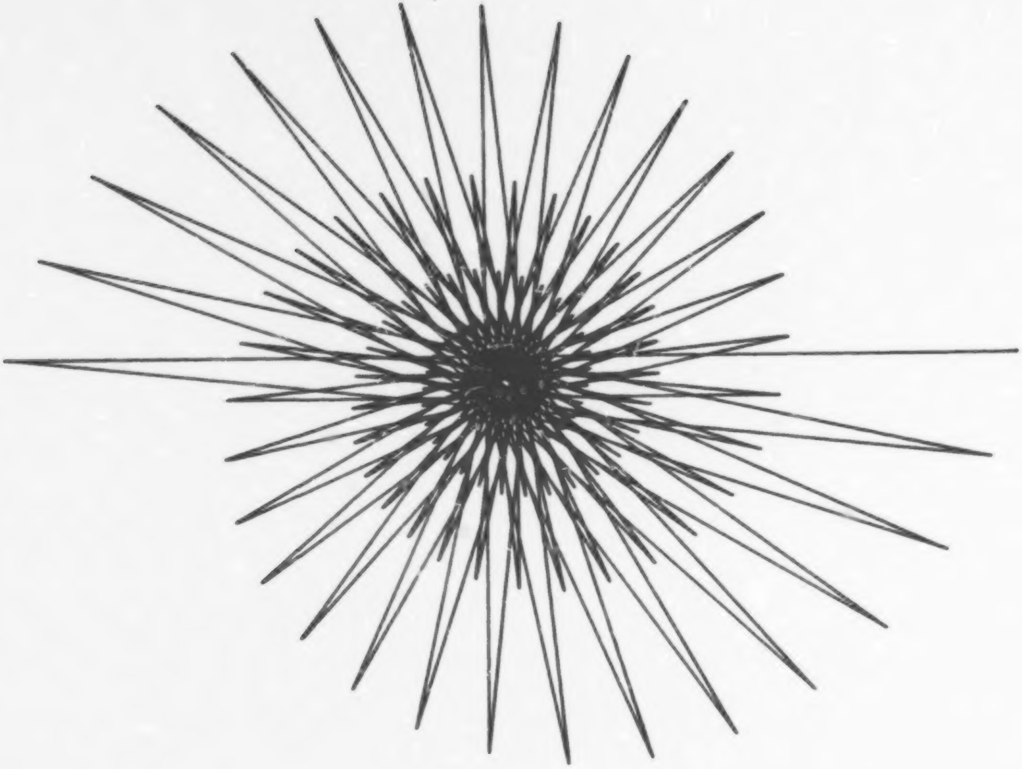


Figure 2. Inward-spiralling design.

LETTERS

by Walter Bilofsky
Bolt Beranek and Newman Inc.

Summary

LETTERS artistically inscribes a message of your choice on the display screen.

Calling Sequence

```
RAC PUBLIC LETTERS <GO>
CALL LETTERS(message, X-origin, Y-origin,  size,  style)
            <text>   <float>    <float>  <float> <text>
                       or
                     <text>
```

Citation

No citation necessary.

Things to Remember

It is a good idea to use the "EA" command before calling LETTERS.

LETTERS works *only* on display terminals.

The public procedure LETTERS draws fancy letters in Old English or Caslon font.

Using LETTERS

To use LETTERS, type the command

```
CALL LETTERS(message, X-origin, Y-origin,
             size, style)
```

where message is the TEXT you want inscribed. X-origin is *either* the FLOAT X screen-coordinate of where you want the display to start *or* the TEXT 'CENTER'. (The screen's X coordinates are -500 to +500; the Y coordinates are -390 to +390.) If you specify 'CENTER', the line is automatically centered on the screen.

Y-origin is the FLOAT Y screen-coordinate of where you want the display to start.

Size is a FLOAT number that determines how large the letters will be. For example, letters of size = 6 are twice as large as letters of size = 3.

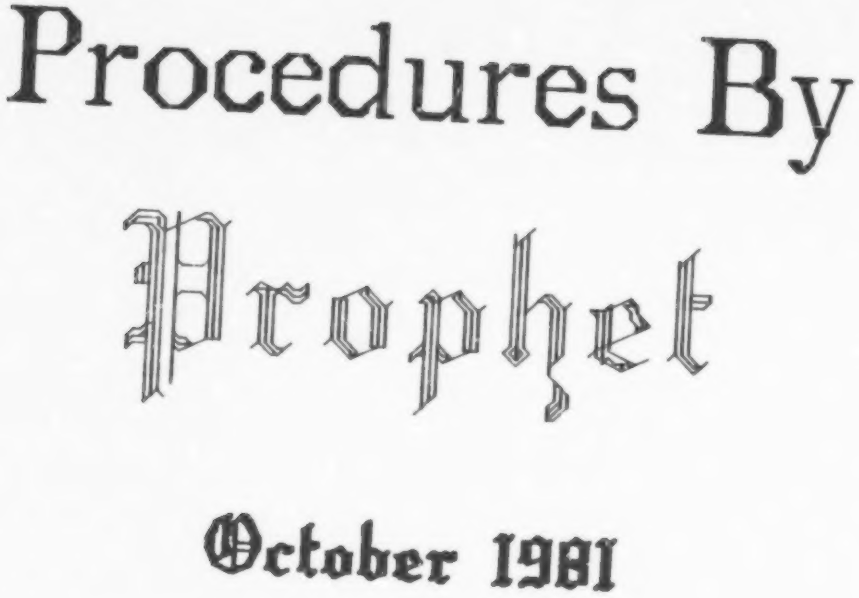
If you specify a negative size, the letters are not filled in.

The last argument, style, is a TEXT that can have either of two values. If style = 'ENGLISH', the procedure uses Old English font; if style = 'CASLON', the lettering is done in Caslon typeface.

Carriage returns and spaces are permitted in the message you want drawn. You can obtain lower case letters by including a backslash character ("\"). (On most terminal keyboards this is SHIFT-L.) Using a backslash causes the procedure to draw all succeeding letters up to a carriage return or to an upper case lock as lower case letters. The character for 'upper case lock' is the slash ('/'), which cancels the effects of a backslash. See the example below for illustration of the use of the slash and backslash.

Consult Figure 1 for an example of the use of LETTERS.

```
>RAC PUBLIC LETTERS<
>CALL LETTERS('P\ROCEDURES   /B\Y','CENTER',120.,4.,'CASLON')<
>CALL LETTERS('P\ROPHET','CENTER',-100.,-8.,'ENGLISH')<
>CALL LETTERS('O\CTOBER   1981','CENTER',-300.,2.5,'ENGLISH')<
```



Procedures By

Prophet

October 1981

Figure 1. Inscribing a message using LETTERS.

7. Appendices

7. Appendices

A. Suggested Guidelines for Writing a Public Procedure . . . A-1

B. Documenting a Public Procedure B-1

C. Using the Public Table "PUBSAVED" to Develop a Public Procedure C-1

D. Incorporating Public Procedures into your PL/PROPHET Programs D-1

App

SUGGESTED GUIDELINES FO

This section is designed to help suggestions offered here will h easy to maintain.

Entries

Try to have "quiet" entr

The most imp write a pub a wide range This means easy for new tedious or v

You should c with your design phas provide two

- the pri for eac

- a "qui specify

Principal Entry

The principa for the P arguments a a "friendl user respon

For instanc program sh empty. If should let opportunity

If your pro should che column real should tel This correc user encou

ndix A

WRITING A PUBLIC PROCEDURE

ou write a public procedure. The
lp make your procedure easy to use and

principal interactive entry and a

rtant point to bear in mind when you
ic procedure is that it may be used by
of users in a variety of applications.
hat you will want your procedure to be
users to understand, but not too
rbose for experienced users.

nsider how various users will interact
rocedure when you are in the early
. One successful approach is to
oints of entry:

cipal entry, which prompts the user
required piece of information

t" entry, which uses arguments to
the necessary information

entry, with its dialogue, is useful
OPHET novice. It should require as few
possible and interact with the user in
" manner. The program should check all
es to see that they make sense.

, if the user specifies a table, your
uld check to see that the table is not
e table is empty, then the procedure
the user know and provide another
to specify a table.

ram asks for a row or column number, it
k to see that the indicated row or
y exists. If it does not, the program
the user and ask for a new response.
ive interaction is preferable to the
ering an "out of bounds" abort.

PL/PROPHET offers many built-in system procedures (e.g., %GETBOUNDEDFIXED) that enable you to write friendly, helpful interactive programs.

"Quiet" Entry

A "quiet" entry is desirable because it allows users to incorporate your procedure into procedures they are writing. Quiet entries also lend themselves to batch jobs, for which all user responses must be pre-specified.

The quiet entry should be functionally equivalent to the principal interactive entry, but should not ask any questions or present any results. Quiet entries usually require arguments corresponding to the answers to the questions in the interactive entry.

Simple results may be directly returned from the procedure, as in the public procedure TRAP. Several complicated results may be "returned" by setting arguments equal to them, as in the FITDIFFQT entry in DIFFEQ.

Naming Procedures

Use short, mnemonic names and stick to conventions.

Try to give your procedure and its entry short names that describe themselves.

Remember not to start procedure or entry names with "%", because this symbol is used to designate system procedures that do not have to be restored and compiled.

To name a quiet entry, add the suffix QT to the name of the main procedure.

Extraneous Files

Delete extraneous files.

If your procedure writes files (e.g., if it calls MLAB), the files should be deleted when your procedure is finished with them. For example, to delete a file named MLASAV.DAT, use the command

```
DELETE FILE 'MLASAV.DAT' <GO>
```

Efficiency

Consider space and time efficiencies.

Complex data objects such as graphs, molecules, and arrays reside in core and may require a large portion of your workarea. Therefore, it is good practice to CLEAR such variables when they are no longer needed.

Pre-allocate arrays and tables that your procedure uses.

PARE tables when your procedure is finished with them.

See the section on time and space efficiencies in the *User's Guide to PL/PROPHET* for other suggestions.

Documentation

Document the definition of your procedure clearly.

It is good practice to write your procedure so that it is easy to understand. Remember that other users might want to modify your procedure to suit their own requirements.

Use helpful comments generously. For example, consider the following comments (the remarks enclosed in the special "comments" symbols, /* */):

```
NSUM = 0; /* set NSUM to 0 */
NSUM = 0; /* initialize sum of weights */
```

The first comment describes *what* is being done, but is really superfluous to the assignment statement. The second comment describes *why* the assignment takes place and is really more helpful.

Structured Programming

Practice structured programming techniques.

"Structured programming" refers to a programming methodology that results in programs that are easy to understand and hence easy to maintain. After you write a public procedure, it is likely that you will need to modify it. (After all, there are many creative PROPHET users who might use your program and suggest some desirable enhancements.) If you wrote your program using structured programming techniques, these modifications will be much easier to make.

Help

Use the PROPHET line (617) 491-7128 for help.

The BBN staff will be happy to answer any of your programming and design questions.

Appendix B

DOCUMENTING A PUBLIC PROCEDURE

This section offers several suggestions on how to document your procedure.

Documentation Format

You should document your public procedures in a format consistent with the procedure descriptions in this manual. Begin with a title page that gives the name of the procedure (the principal entry), the author, and the author's site. Follow this heading with a "summary box" that serves as a quick reference to the procedure. See the Introduction to this notebook for a description of the contents of a summary box.

Most of the documentation should be a detailed description of both the theory behind the procedure and the actual operation of the procedure. Be sure to describe all available entries and explain when to use each entry. Provide references where appropriate.

Providing Examples

Actual examples of using the procedure are very helpful. If you send us documentation along with hard copies of examples, we will be happy to distribute them to interested users. Please be sure that the examples can easily be reproduced. If your procedure requires tables, you should make public versions of example tables and archive them. This way, other users can duplicate the examples and use them to learn how to use your procedure.

Making Your Documentation Available

You can use the PROPHET text editor to compose the procedure documentation on-line. The name of the documentation text should be the principal name of the procedure followed by the suffix DOC. Save the documentation in the PUBLIC SAVED table or the public table PUBSAVED (see Appendix C). Finally, set the notes of the procedure to a short summary of how to obtain this on-line documentation.

You are encouraged to send a message to all PROPHET users (e.g., SEND MESSAGE T TO 'SYSTEM' <GO>) when your procedure is publically available. This message should include a short description of the procedure plus instructions for how to call it and how to obtain documentation for it. Be sure to include your PROPHET mailing address or phone number.

Appendix C

USING THE PUBLIC TABLE "PUBSAVED" TO DEVELOP A PUBLIC PROCEDURE

All procedures in the PUBLIC SAVED table have been thoroughly tested. To maintain the integrity of this table, we strongly suggest that you save your procedure in the public table PUBSAVED before you save it in the PUBLIC SAVED table. Historically, PUBSAVED has become known as the "Public Scratch Area". This table is a convenient place to save procedures that you are developing, along with their documentation.

Getting Feedback

Saving your procedure in PUBSAVED enables users to provide you with feedback. The PROPHET community includes a diverse range of scientists who can provide valuable suggestions and comments.

Feel free to send a message to all users announcing when the procedure becomes available from PUBSAVED. Remind users that your procedure is under development and that you would appreciate their feedback. Allow an ample period of time for users to respond (about 6 months).

Moving to the PUBLIC SAVED Table

The number of times your procedure was restored and compiled is automatically recorded in column 5 of PUBSAVED. If there has been sufficient interest in your procedure, it should be moved to the PUBLIC SAVED table. Please notify us when you think your procedure should be moved to the PUBLIC SAVED table, and we will be glad to help.

Appendix D

INCORPORATING PUBLIC PROCEDURES INTO YOUR PL/PROPHET PROGRAMS

Using a globally defined procedure is a good, convenient programming practice in PL/PROPHET. It helps you avoid having to "re-invent the wheel". If you need an internal procedure that has already been written as a public procedure and tested, you can simply incorporate it within your program by declaring it to be a global.

System Procedures

The PROPHET programming environment abounds with hundreds of debugged and tested specialized procedures that you may use in your PL/PROPHET programs. Many of these are the built-in "system procedures"; by convention, all system procedures begin with a percent symbol, % (e.g., %T2A and %LASTROW).

Using system procedures is similar to using global procedures because they are defined outside your program. However, because they are built into PROPHET, you need not explicitly declare them in your programs. PROPHET already knows that they are procedures and understands how to compile references to them.

Using Public Procedures

You are also encouraged to take advantage of the many public procedures that are available. Most of them have quiet entries that permit you to incorporate them within your own procedures in a clean, straightforward manner.

Because public procedures are NOT built in, as are system procedures, you must declare, restore, and compile them before using them. The following example illustrates how to construct a PL/PROPHET program that incorporates the public procedure TRAP.

```
PROCEDURE;
          -
          -

      /* local declarations, e.g., "DECLARE DATATAB TABLE;" */
          -
          -
      DECLARE TRAP PROCEDURE! /* global definition of TRAP procedure */
          -
          -
          -
      IF EMPTY(NOEVAL TRAP) THEN   /* RACing TRAP into the workarea */
      RAC PUBLIC TRAP NODEF;
          -
          -
          -
      TYPE 'AREA = ', TRAP(DATATAB); /* Using the public proc TRAP */
          -
          -
          -
END;    /* of procedure */
```

You should note the following about this example:

- Because TRAP is defined globally, its declaration must end with an exclamation point (!). Identifiers that are defined locally to your procedure end with a semicolon (;).

- Before you try to RAC (restore and compile) TRAP, test to see whether it has already been RAC'ed simply by testing whether the global variable TRAP is EMPTY. (Remember that NOEVAL is required with the name of a procedure when you do not wish to call the procedure.)

  If you did not perform this initial test, then every time you called your procedure, it would RAC TRAP. After the first call, TRAP would already be restored and compiled, so each subsequent RAC would result in the following message from PROPHET:

  ```
  WARNING:  RESTORED VARIABLE WILL SUPERSEDE
  AN EXISTING VARIABLE.  DO YOU WISH TO
  CONTINUE?
  ```

Clearly, you do not want to answer this question each time you use your procedure. Thus, your program should include the "EMPTY test".

- The NODEF option indicates that only the compiled code -- not the definition of the procedure -- should be restored. Some public procedure definitions are quite large and can clutter your workarea. Thus, it is good practice to use the NODEF option unless you need the definition of a procedure.
- After TRAP has been restored and compiled, it can be called within your program.
- To use the quiet entry TRAPQT, instead of calling TRAP in your procedure, you would only need to add the global declaration of TRAPQT:

  ```
  DECLARE (TRAP,TRAPQT) PROCEDURE!
  ```

  Even though TRAPQT is an "entry" to the procedure TRAP, it is still functionally a procedure. However, you would still use the same "RAC PUBLIC TRAP" command because the procedure is only known by its principal name in the PUBLIC SAVED table. Once you restore and compile TRAP, you can use the quiet entry TRAPQT throughout your program.

8. Index

8. Index

INDEX

Page

ABSORB 4-35
ABSORBQT 4-35, 4-37
AGONIST 4-7
AGONIST2 4-7, 4-12
ANTAGONIST 4-17
ANTAGONIST2 4-17, 4-20
AREA 3-23
AREARA 3-23
AREATAB 3-23

BARGRAPH 2-47
BARGRAPHCONST 2-47, 2-50
BARGRAPHQT 2-47, 2-49

CELLSIM 4-147
COMPNAMED 5-1
COMPSAVED 5-1
COMPSHARED 5-1

D3DQT 2-29, 2-41
DATA2TBL 4-89, 4-110
DELETEARCHIVED 5-3, 5-9
DELETESHARARCHIVED 5-9
DELETETABS 1-21
DIFFEQ 4-43
DIGITIZE 2-11
DIGITIZEPTS 2-11, 2-15
DIGITIZEPTSQK 2-11, 2-18
DIGITIZEQK 2-11, 2-16
DRAW3D 2-29
DRUGAUC 4-70, 4-81
DRUGAUCQT 4-81, 4-86
DRUGFUN 4-63, 4-81, 4-88, 4-127, 4-131, 4-139
DRUGFUNQT 4-63
DTOGPOINT 2-1
DTOGPOINTQT 2-1, 2-2

EDCOL 1-23
EDITMDL 4-89, 4-110
EDITPOINTS 2-3
EIGEN 3-5

FILLTABS 1-7
FINDARCHIVED 5-3, 5-9
FINDSHARARCHIVED 5-3
FIT2PARM 4-89, 4-111
FITANDINTDIFF 4-43, 4-59
FITANDINTDIFFQT 4-43, 4-60
FITDIFF 4-43, 4-46
FITDIFFQT 4-43, 4-59

Page

GOFIT 4-71, 4-127
GOFITQT 4-127, 4-129
GRAFIT 4-70, 4-131
GRAFITQT 4-131, 4-132
GRAPH3D 2-21

HAPPY 6-1

INTDIFF 4-43, 4-55
INTDIFFQT 4-43, 4-59
INTEGRAL 3-23
INTEGRALRA 3-23
INTEGRALTAB 3-23
INTERPOLATE 3-9
INTERPOLATEQT 3-9, 3-12

LETTERS 6-9
LOGISTIC 4-27
LOGISTIC2 4-27, 4-33
LOOPMDL 4-89, 4-111

MAKEFORMATTAB 5-13, 5-19
MAKETABNAMES 1-1
MAKETABNAMESQT 1-1, 1-4
MAKETABS 1-5
MAKETABSQT 1-5, 1-6
MKMDLTAB 4-89, 4-96
MKMODEL 4-89, 4-127, 4-131, 4-139
MKMODELQT 4-89, 4-104
MKPARMTAB 4-89, 4-101
MLAB2P 5-35
MLAB2PQT 5-35, 5-36
MODELTEST 4-139
MULT3D 2-43
MULTINTERPOLATE 3-9, 3-11
MULTINTERPOLATEQT 3-9, 3-12

P2BMDP 5-39
P2MLAB 5-31
P2MLABQT 5-31, 5-32
PLANIM 3-21
PLANIMQT 3-21, 3-22
POLYGON 6-3
POOLTABS 1-11
POOLTABSQT 1-11, 1-13
PRINTTABS 1-9
PROBIT 4-1

QG 4-131, 4-134
QMDL 4-89, 4-106

RBCELLSIM 4-147, 4-157
READFILE 5-13, 5-19
READTEXT 5-29

Page

REDRAW3D 2-29, 2-41
ROOT 3-1

SETOPTION 4-95
SMOOTH 3-17
SPIRO 6-5
STEPAREA 3-31
STEPAREAQT 3-31, 3-32

TABLE2FILE 1-27
TABLE2FILEQT 1-27, 1-29
TRAP 3-27
TRAPQT 3-27, 3-29

XFROMY 2-53, 2-55
XFROMYPOLY 2-53, 2-56
XFROMYQT 2-53, 2-57

best copy available